

Design and Implementation of Wireless Sensor Networks Using LoRaWAN, MQTT and Cloud Computing



Master Thesis

submitted for the degree of
Master of Science in Mechatronics
by

Vishu Sharma

Matriculation Nr: 1523731

Faculty IV
Chair for Embedded Systems

May 2023

First Supervisor : Prof. Dr. Roman Obermaisser
Second Supervisor : Dr. Daniel Onwuchekwa

The objective of this thesis is to create small-scale Wireless Sensor Networks (WSNs) that leverage LoRaWAN technology, MQTT protocol, and cloud computing as a proof of concept for an IoT platform. These WSNs are intended to be long-range, low-power networks that are scalable, cost-effective, and relatively easy to deploy. The work begins with a literature review of the current state-of-the-art Wireless Sensor Networks (WSNs) based on technologies like WiFi, Bluetooth, ZigBee, and Low Power Wide Area Networks (LPWANs). LPWAN technologies are commonly used for long-range networks. This review revealed that LoRaWAN technology has several advantages over other LPWAN technologies such as SigFox and Narrow Band IoT (NB-IoT) in terms of cost-effectiveness and ease of network deployment. MQTT protocol was chosen for its lightweight and data-centric approach in comparison to HTTP's document-centric approach. Cloud computing was chosen for its scalability, security, and easy integration with existing infrastructure.

The research work investigates a number of important issues concerning the challenges involved in integrating and optimizing the WSNs based on LoRaWAN, MQTT, and cloud computing systems. One of the key issues explored during the research was where to perform certain tasks related to LoRaWAN-based wireless sensor networks, such as decoding sensor data using a decoding script. This can be done either in the LoRaWAN gateway, in resource-constrained Single Board Computers (SBCs) at the edge, or in cloud/local servers. The research also explores the optimal MQTT architecture for sensor applications running in LoRaWAN gateways such that it's compatible with the back-end computing infrastructure. Additionally, the research examines the current limitations of LoRaWAN-compatible network devices, particularly with regard to ease of network deployment, configuration tools, and systems integration.

Key Words: WSN, LoRaWAN, MQTT, Sensors, Cloud computing, IoT, Edge computing, system integration, LPWAN, Industry 4.0.

Acknowledgements

I would like to express my gratitude to my colleagues, friends, and advisors who have contributed to the successful completion of my thesis. This thesis was done in cooperation with APS-Tech Group GmbH and Embedded Systems group at Universität Siegen.

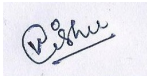
I am immensely grateful to APS-Tech Group GmbH for providing all the necessary material support in the form of equipment such as sensor modules, LoRaWAN gateways, development boards, and servers. I would like to thank the founder of APS-Tech Group Armin Seibold for providing support and encouragement throughout the entire process. Without his generosity and encouragement, this research would not have been possible.

I would like to thank my colleagues Christian Ngameni, Jasper Hatilima, Issac Hatilima, and Patrick Vaas who have provided guidance and support throughout my research journey. Their technical expertise and guidance in working with software development, MQTT, cloud computing, IoT, and electronics were crucial to the success of my research. When I started my thesis I had limited knowledge in these areas but with their guidance, I developed competence and improved my skills.

I extend my sincere gratitude to Prof. Dr.-Ing Roman Obermaisser and Dr.-Ing. Daniel Onwuchekwa for supervising this thesis and providing the freedom to pursue my academic interests. The opportunity to conduct research in a supportive environment was a privilege.

Declaration of Independent Work

I, Vishu Sharma, hereby declare that the work done in this Master thesis titled "Design and Implementation of Wireless Sensor Networks Using LoRaWAN, MQTT and Cloud Computing " is my own work and it has not been previously submitted for any degree or other purposes. Furthermore, all sources that assisted me throughout my Master's thesis work, including other researchers' work published, have been fully cited.



.....

Siegen, 17.05.2023

1	Introduction	1
1.1	Motivation	1
1.2	Introduction to LPWAN	1
1.3	Comparison of LPWAN with existing technologies	2
1.3.1	Range and power consumption	2
1.3.2	Data rate and payload size	3
1.3.3	Cost and ease of network deployment	3
1.4	Applications of LPWAN networks	3
1.4.1	Industrial automation	3
1.4.2	Smart cities	3
1.4.3	Smart agriculture	4
1.4.4	Supply chain management and logistics	4
1.4.5	Environment	4
1.5	Limitations and challenges with LPWAN	4
1.6	LoRaWAN compared to other LPWAN Technologies	5
2	LoRa: PHY Layer	7
2.1	Introduction to LoRa	7
2.1.1	Key features of LoRa	8
2.1.2	ISM bands for LoRa	8

2.2	LoRa modulation	8
2.3	LoRa PHY data packets	9
2.4	LoRa modulation design parameters	9
2.5	Design Equations for LoRa	11
2.6	LoRa Devices	14
2.6.1	Sensor Nodes	14
2.6.2	Gateways	16
2.6.3	Servers	17
3	LoRaWAN: MAC Layer	19
3.1	Introduction to LoRaWAN	19
3.2	LoRaWAN Network	19
3.2.1	Information flow and security in LoRaWAN	20
3.2.2	Parameters for configuring LoRaWAN devices	21
3.3	Device activation in LoRaWAN	22
3.3.1	Activation by Personalization	22
3.3.2	Over The Air Activation	22
3.4	Device class in LoRaWAN	23
3.5	LoRaWAN MAC Message Format	25
3.6	Adaptive Data Rate	25
3.7	LoRaWAN Technical Specifications	26
3.7.1	LoRa Data Rates	26
3.7.2	Relationship between SF, Time on Air and Receiver Sensitivity . . .	27
4	MQTT	28
4.1	Introduction to MQTT	28
4.2	MQTT communication architecture	28
4.3	MQTT control Packets:	30
4.4	What makes MQTT suitable for IoT applications?	30
5	Overview of System Architecture and Network Implementation	32

6	Initial Experiments with LoRa PHY and LoRaWAN MAC	34
6.1	Tests with LoRa PHY layer	34
6.1.1	Objective and description of tests:	34
6.1.2	Transmitting node and receiver module	34
6.1.3	Software programming of development boards	35
6.1.4	Observation and results of tests	38
6.1.5	Limitations of the tests with Lora PHY	39
6.2	Tests with LoRaWAN MAC Layer	40
6.2.1	Node-RED:	41
6.2.2	Payload decoding and modification	42
7	WSN Implementation using LoRaWAN, MQTT and Local Servers	44
7.1	Network Architecture	44
7.1.1	Sensor Nodes	44
7.1.2	Gateway: As an MQTT Client	45
7.1.3	MQTT Servers	46
7.1.4	MQTT Subscribing Clients	46
7.1.5	Information flow in the network	46
7.2	Data Logging System of network	47
7.3	Network security implementation	49
8	Results and Analysis	51
8.1	EDA of temperature humidity sensors	51
8.2	EDA of data collected from ultrasonic distance sensors	53
8.3	Analysis	57
8.3.1	Challenges with message payload decoding	57
8.3.2	Challenges with scaling	57
9	Prototype Network with Cloud Server	59
9.1	Microsoft Azure cloud services	60
9.2	Network I	61
9.3	Network II	62

10 Summary and Conclusion	66
10.1 Challenges and scope of future work	66
10.1.1 Challenges with the LoRaWAN gateways	66
10.1.2 Challenges with LoRaWAN sensor nodes	67
10.2 Challenges with MQTT and cloud integration	68
10.3 Recent advances in LoRa/LoRaWAN technology	68
10.4 Design principles	69
A LoRa Communication	70
A.1 Terminology and concepts in LoRa Communication	70
B LoRaWAN Network Equipment and Development Tools	72
B.1 OEM and Service providers	72
B.2 LoRa Development Boards and Modules	72
B.3 Radio Modules and LoRa chips	73
B.4 Gateways and Nodes	73

List of Figures

1.1	Comparison of wireless communication technologies on range vs power consumption	2
2.1	LoRaWAN protocol stack	7
2.2	Structure of LoRa data packet	9
2.3	LoRa Sensor Nodes	14
2.4	Schematic showing internal architecture of LoRa sensor node	15
2.5	LoRaWAN gateway internal architecture	16
2.6	Web GUI of LoRaWAN gateway	17
3.1	LoRaWAN network topology	20
3.2	Information Flow in a LoRaWAN Network	21
3.3	Class A device receive windows	23
3.4	Class B device receive windows	24
3.5	Class C device receive windows	24
3.6	LoRaWAN MAC message format	25
4.1	MQTT network architecture	29
5.1	Abstraction of simplified network architecture	32
6.1	Node to node communication system setup	35
6.2	LoRa development board programmed as transmitting node	36

6.3	Heltec LoRa development board	36
6.4	LoRa development board programmed as receiver node	36
6.5	Logged sensor data from the transmitting node	37
6.6	Logged sensor data from the receiver module	38
6.7	System abstraction and information flow	40
6.8	Message Flow diagram implemented in LoRaWAN gateway using Node-RED	41
6.9	Payload decoding and modification using Node-Red	42
6.10	Payload decoding logic for temperature humidity sensor	42
7.1	System architecture of network with local MQTT servers	44
7.2	Temperature-humidity and ultrasonic distance sensor nodes	45
7.3	MQTT client applications inside gateway and flow of information in the network	47
7.4	Raw message payload from ultrasonic distance sensor	48
7.5	Node-RED flow for sensor payload decoding and data logging	48
7.6	Data logged from ultrasonic distance sensor after modification	49
7.7	Network Security with local MQTT server and clients	50
8.1	Data collected from temperature humidity sensor rht-01	52
8.2	Temperature humidity correlation	52
8.3	Data collected from temperature humidity sensor rht-02	52
8.4	LoRa frequency channel utilization for sensor rht-02	53
8.5	LoRa SNR variation for sensor rht-02	54
8.6	RSSI variation for sensor rht-02	54
8.7	Data collected from ultrasonic distance sensor uds-01	55
8.8	LoRa frequency channel utilization for sensor uds-01	55
8.9	LoRa SNR variation for sensor uds-01	56
8.10	RSSI variation for sensor uds-01	56
9.1	Network architecture with Azure back-end	60
9.2	Recorded data from Network I	62
9.3	Network I architecture	63

9.4	Network II architecture	64
9.5	Obodo Monitoring Dashboard	65

1.1 Motivation

The motivation behind this thesis was to address the challenges involved in creating small-scale Wireless Sensor Networks (WSNs) using LoRaWAN technology, MQTT protocol, and cloud computing systems. The research work done during this thesis contributes to the field of low-power long-range WSNs by providing valuable insights into the optimal network architecture, challenges in system integration, and challenges of doing computation at the edge vs cloud for LoRaWAN-based networks. The findings of this study will help in the development of low-power, long-range, scalable, and cost-effective WSNs that can be easily integrated with cloud computing infrastructure.

This chapter discusses the unique features of LPWAN technology and compares it with existing wireless communication technologies like WiFi, Bluetooth, and cellular networks on key design parameters. Applications of LPWAN networks in IoT were discussed briefly. LoRaWAN is compared with other LPWAN technologies like SigFox and NB-IoT.

1.2 Introduction to LPWAN

Low Power Wide Area Network (LPWAN) is a type of wireless communication network suitable for sending small data packets over long distances. End devices used in LPWAN consume minimal power and these devices can function on small batteries for years. LPWAN technologies have a variety of applications in smart cities, industrial automation, healthcare, and agriculture. There are 3-4 main competing technologies in the LPWAN space. Each technology has some unique features that determine the feasibility of application for the specific use cases [5]. Prominent LPWAN technologies are listed below:

- Long range Wide Area Network (LoRaWAN)
- Narrow Band IoT (NB-IoT)

- SigFox
- LTE-M

1.3 Comparison of LPWAN with existing technologies

LPWAN has some unique features compared to other existing wireless communication technologies like WiFi, Bluetooth, and Cellular networks which makes it suitable for Internet of Things (IoT) and machine to machine communication (M2M) applications. These key features are discussed below:

1.3.1 Range and power consumption

Technologies like WiFi and Bluetooth are only suitable for transmitting data at short-range distances (10-100m). In cellular networks with 3G/4G data can be transmitted at longer range (5-15 km) but the power consumption of end devices (like smart phones) as well as base stations/cell site is quite high. LPWAN technologies can provide long range communication (2-50 Km) at very low power consumption rate. Devices using LPWAN can run on small batteries for 5-10 years! These features makes LPWAN suitable for sending data from remote sensor nodes running on batteries. See figure 1.1 for a qualitative comparison between different Wireless Communication Technologies (WCTs) on the basis of range vs power consumption and data rates.

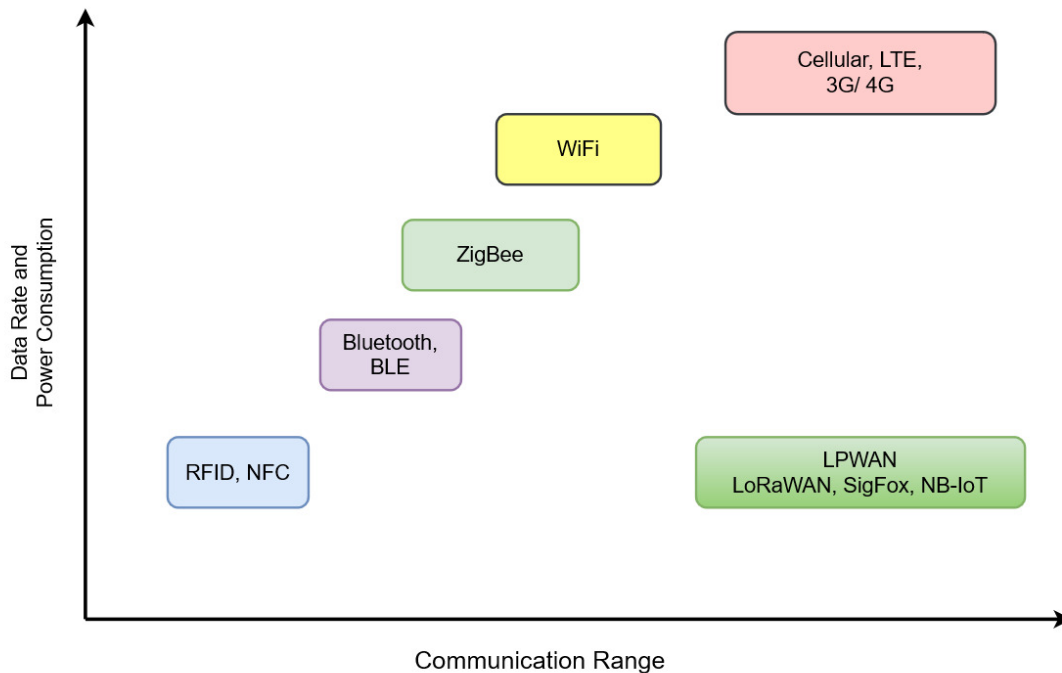


Figure 1.1: Comparison of wireless communication technologies on range vs power consumption
Source: Based on signal ranges of 5G, LPWAN and other wireless technologies

1.3.2 Data rate and payload size

LPWAN technologies have relatively low data rates compared to WiFi, Bluetooth, and cellular networks. For example, SigFox, LoRaWAN, and NB-IoT have data rates of up to 100 bps, 50 kbps, and 200 kbps respectively. Whereas WiFi and cellular networks can transmit data at much higher rates of a few hundred Mbps. The maximum payload size is also limited in LPWAN networks. For example, SigFox, LoRaWAN, and NB-IoT have a maximum payload size of 12 bytes, 243 bytes, and 1600 bytes. For most IoT applications like sensor networks, a lower data rate and small payload size are sufficient. [9]

1.3.3 Cost and ease of network deployment

LPWAN networks are significantly less expensive to deploy and operate compared to the cellular networks. The devices like end-nodes, gateways and base stations used in LPWAN networks are relatively cheaper than their counterparts in cellular network. LPWAN networks like SigFox and LoRaWAN operates in the unlicensed frequency bands. They do not require sophisticated infrastructure and regulatory approvals. For example nodes and gateways for LoRaWAN network costs in the range of 10-50 € and 100-1200 € respectively. This makes the deployment of experimental networks faster and cheaper. Note that base stations for NB-IoT are expensive and costs around €10,000-15,000 . Also NB-IoT network requires costly frequency spectrum to operate [18].

1.4 Applications of LPWAN networks

LPWAN is expected to play a significant role in the development of smart cities, Industry 4.0, smart agriculture, and other Internet of Things (IoT) applications. Some examples of sensor data that can be sent by using LPWAN Network (specially LoRaWAN) are temperature, humidity, pressure, soil moisture, CO₂ and particulate matter concentration(PM 2.5), etc. Major application areas and use cases for LPWAN based IoT systems are listed below [5]:

1.4.1 Industrial automation

- Temperature and humidity and indoor air quality monitoring in pharmaceutical and food processing industry
- Fluid level monitoring in tanks for industries like chemical, textile and petroleum
- Preventive maintenance in machines
- M2M communication applications for automation

1.4.2 Smart cities

- Smart utilities: Water level monitoring, sewage monitoring
- Health: air quality index and particulate matter monitoring
- City waste management and garbage collection
- Smart buildings: Ambience monitoring, CO₂ monitoring, building occupancy counter.

- Parking occupancy by smart parking sensors

1.4.3 Smart agriculture

- Monitoring crop growing conditions like light intensity
- Temperature and humidity monitoring in smart greenhouses
- Toxic gas level monitoring in compost and livestock enclosures
- Soil moisture level and pH level monitoring

1.4.4 Supply chain management and logistics

- Asset tracking
- Cold chain monitoring
- Automating stock checking

1.4.5 Environment

- Geo location: Wildlife monitoring for endangered species
- Pollution level monitoring
- Weather monitoring
- Collecting temperature data in remote places for climate change monitoring
- River flood monitoring

1.5 Limitations and challenges with LPWAN

Limited bandwidth, data rate and packet size: The desirable features of long range and long battery life in LPWAN comes with a lot of trade-offs. For example, Band Width (BW), data rates (bits/sec) and the maximum payload size (data packet size in bytes) are limited in LPWAN. LPWAN is not suitable for sending and receiving large data files like images, videos or voice calls that requires high speed data transfer. LPWAN technology is also not suitable for real time applications in industrial control or mission critical applications which requires low latency and high reliability. For example the latency in LoRaWAN and NB-IoT networks is quite high (up to tens of seconds) whereas industrial control applications require latency of less than few hundred ms. Although NB-IoT has lower latency as compared to LoRaWAN.

Interference: SigFox and LoRaWAN operates on unlicensed ISM bands. This makes them susceptible to other devices operating in the same frequency band and effects the reliability of the network. In the dense urban areas buildings can block the radio signals. Consequently, the communication range between gateways and end-devices is significantly reduced. For example LoRaWAN gateways can have range of 10-15 Km in rural areas when a clear line of sight is present between gateway and the end device. This range reduces to 2-5 Km in dense urban areas.

Compatibility and network coverage: LPWAN network technologies are not always compatible with existing communication technologies and also with each other. For example LPWAN end devices from SigFox and LoRaWAN cannot use cellular networks for sending data. LPWAN networks have limited network coverage as compared to cellular networks. They also do not support roaming of end devices between networks. This makes it difficult for end devices to switch between different networks.

Security and standardization: Security is a big challenge for the future of LPWAN-based IoT systems. The security challenge is further compounded by the large number of devices in the network. Currently, LPWAN networks lack in the security as compared to the existing cellular networks. There are security vulnerabilities at the hardware level, firmware level, and network level. Standards for LPWAN are still under development. The 3rd Generation Partnership Project (3GPP) is responsible for NB-IoT, and the LoRa Alliance is in charge of developing the LoRaWAN standard. There is a lack of standardization in LoRaWAN at various levels - end devices, gateways, servers, software, etc.

Power Consumption: The power consumption of end devices in LPWAN networks is significantly less compared to the WiFi, Bluetooth and cellular networks. But LPWAN based IoT systems are supposed to have a very large number of end devices in the network. The collective energy consumption of these large number of end devices could be a lot even though each end device consumes very less energy individually. The cost of battery replacement becomes huge for large scale network deployment. This problem need to be addressed with energy harvesting technologies like from solar energy. However this is not very common with most of the commercial sensor nodes available in the market.

1.6 LoRaWAN compared to other LPWAN Technologies

Each LPWAN technology offers some unique features suitable for different applications. LoRaWAN has been selected over other LPWAN technologies in this research due to many reasons. These are summarized below:

- **Open standard:** LoRaWAN is an open standard. The developer ecosystem and the resources available to build private networks is quite sufficient. This makes the prototyping relatively easy as compared to networks based on Sigfox and NB-IoT.
- **Cost:** The network equipment costs and the operating costs of running the LoRaWAN network is very low. NB-IoT gateways/base stations costs significantly more. Sigfox has monthly operating costs associated with each end device. This makes LoRaWAN a more affordable choice for small scale network deployments.
- **Ease of network deployment:** LoRaWAN is a mature technology as compared to NB-IoT. As LoRaWAN operates in unlicensed ISM bands, the private network deployment is relatively easy.
- **Range and power consumption:** LoRaWAN networks provide better coverage at very low power consumption as compared to NB-IoT. This makes LoRaWAN more suitable for applications in smart cities and smart agriculture. Although the latency offered by NB-IoT is better. NB-IoT is better suited for applications requiring low latency such as smart metering and industrial automation. For more detailed comparison of LPWAN technologies on the basis of energy consumption see [28].

A more detailed comparison of LPWAN technologies on various parameters has been investigated by previous studies [10] [18]. The following chapters provide a comprehensive

overview of the working of LoRaWAN at both physical layer (LoRa PHY) and the media access layer (LoRaWAN MAC) in great detail. These chapters covers LoRaWAN related topics in depth and serves as the theoretical framework on which LoRaWAN based WSNs were designed and implemented during the research.

2.1 Introduction to LoRa

LoRa (Long Range) is an RF modulation technology based on Chirp Spread Spectrum (CSS). The modulation technique used in LoRa is closed-source and patented. LoRa PHY is the physical layer in LoRa/LoRaWAN protocol Stack as shown in figure 2.1 (Source: based on). This is similar to the "bits" or physical layer in the OSI seven-layer model. LoRa enables wireless communication data links over very long distances. This range of (2-15 Km) in LoRa is significantly longer as compared to other wireless communication technologies like WiFi, Bluetooth, and ZigBee. LoRa transceivers are low-power devices that send small data packets (maximum 242 bytes) at low data rates (maximum 50 kbps) in sub-GHz licence-free Industrial Scientific and Medical (ISM) bands. [18] [25].

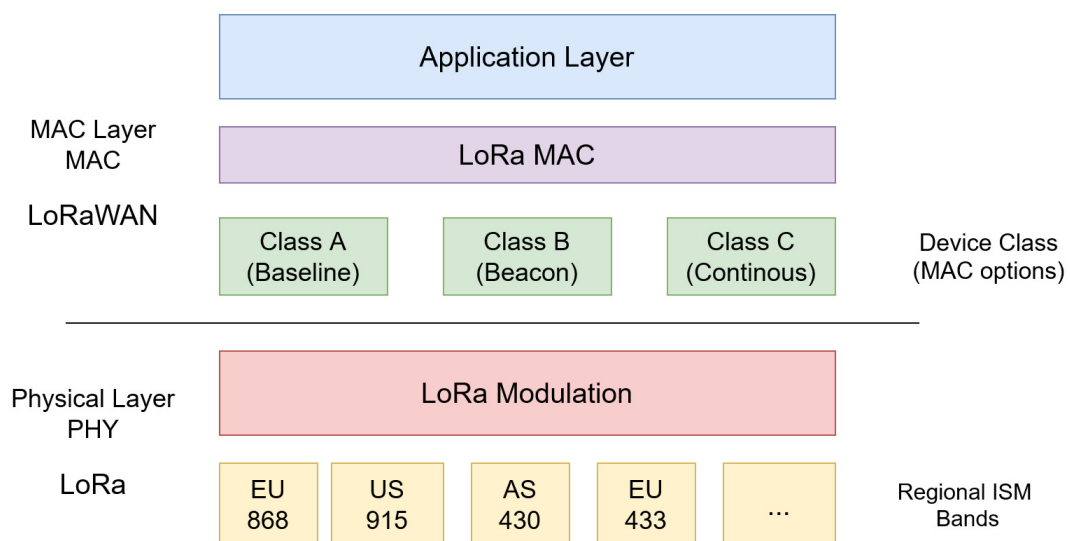


Figure 2.1: LoRaWAN protocol stack

Source: Semtech Corporation: LoRa and LoRaWAN a technical overview

2.1.1 Key features of LoRa

LoRa PHY radio link in the physical layer is the best-effort transmission. It will transmit the data packet but will not give any acknowledgment if the packet has reached its destination or not. LoRa provides cost-effective and energy-efficient wide-area network communication capabilities, which makes it suitable for Internet of Things (IoT) applications in smart cities, agriculture, and industrial automation. Here are some of the key features of LoRa [22]:

- Cost effective due to the use of unlicensed sub-GHz ISM bands.
- High resilience to noise and interference due to the use of Chirp Spread Spectrum (CSS).
- Capacity, payload size, and data rates are low which makes it suitable for transmitting sensor data.
- Long battery life due to very little power consumption.
- Large link budget that enables long-range communication.
- Robust against channel degrading mechanisms like the Doppler effect, multipath, and fading.
- Secured data transmission possible with encryption in MAC layer.

2.1.2 ISM bands for LoRa

LoRa operates in Sub-GHz Industrial, Scientific, and Medical (ISM) radio bands as depicted in figure 2.1. This frequency spectrum is reserved for unlicensed use. These bands can be used by anyone without paying any licensing fee to government and regulatory bodies. LoRa exploits this feature to keep the network operation costs low. These bands are also used by consumer electronic devices like microwave ovens, home WiFi routers, NFC card readers etc. ISM radio bands comes with a trade-off that for no licensing fee there are low data rates and lots of interference from other sources.

Allowed operating frequency spectrum for LoRa varies from region to region. For example, in the European Union the permitted ISM radio band for LoRa communication is 863-870 MHz, in Asia it is 433 MHz and for USA the permitted band is 915 MHz. Use of ISM bands in EU is regulated by European Telecommunication Standards Institute (ETSI). To make fair and effective use of spectrum, ETSI has certain regulations on the maximum transmit power (Tx Power), duty cycle and packet air time. These regulations effects how frequently the messages can be transmitted from sensor nodes using LoRa. More about this is discussed in the section LoRaWAN rules and regulation in next chapter. See ETSI regulations for Short Range Devices (SRD) operating in the frequency range 25 MHz to 1000 MHz [6].

2.2 LoRa modulation

The modulation technique used in LoRa is quite different from digital signal modulation techniques like Frequency Shift Keying (FSK) or Amplitude Shift Keying (ASK). LoRa uses a propriety modulation technique derived from Chirp Spread Spectrum (CSS) modulation.

CSS is suitable for a communication system with a very large number of end nodes. CSS spreads a narrow band signal over a wider channel bandwidth. In CSS, the signal is transmitted in short bursts or chirps. The key idea that LoRa uses to encode information is chirp pulses whose frequency increases and decreases over a certain period. These chirp signals are used as carrier signals. The data signal is modulated on top of this chirp signal. Combined signal (carrier + data) does pseudo-random hopping between a set of allowed frequencies in the ISM bands. LoRa modulation also uses parity bits for error correction. [25] [27]

Direct Sequence Spread Spectrum (DSSS) modulation required the use of a highly accurate and expensive Temperature compensated crystal oscillator (TCXO) for reference clocks. Whereas LoRa chips do not need these expensive TCXO for robust communication due to the use of chirp as a carrier signal. Since LoRa PHY is proprietary, the exact algorithms used to achieve Lora modulation at the radio level are still not known. Many reverse engineering attempts have been tried to deconstruct the mystery of Lora PHY. Lorenzo Vangelista describes LoRa as Frequency Shift Chirp Modulation (FSCM). For a more mathematically rigorous description of LoRa, see [30] [29]

2.3 LoRa PHY data packets

In the LoRa PHY layer, a LoRa data packet for uplink communication has 3-4 main elements depending on the mode of communication. These are shown in figure 2.2

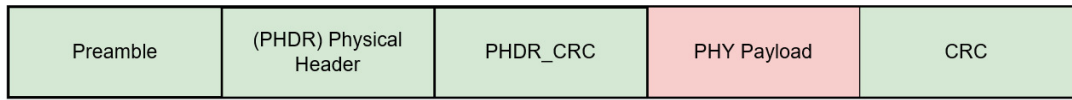


Figure 2.2: Structure of LoRa data packet

Source: Semtech Corporation: LoRa and LoRaWAN a technical overview

- Preamble: All LoRa data packets in PHY layer contains a preamble. Preamble allows the receiver (LoRa gateway) to detect the incoming LoRa packets.
- PHY Header(PHDR): Header is optional, it is only used in explicit mode. Header contains information about payload length, Coding Rate (CR) and information about PHDR Cyclic Redundancy Check (If CRC is present or not?)
- PHY Payload: PHY payload is the most important part of the LoRa data packet. It contains the actual sensor data transmitted. When the LoRaWAN protocol is implemented, the PHY payload also carries MAC information like MAC payload, MAC header, Message Integrity Code (MIC) and device address etc.
- CRC: Cyclic Redundancy Check (sometimes referred to as Payload CRC) is used for error detection in the data packets by using a checksum of the bits transmitted and received. Note that a LoRa uplink packet is different from the downlink packet. In down-link packets CRC is not present.

2.4 LoRa modulation design parameters

While implementing a LoRa radio communication link, there are some important design parameters which needs to be considered. These parameters effects power consumption

rate, data rate, maximum payload size, packet air time, link budget and ultimately the communication range. The most important controllable parameters for LoRa PHY are: Transmit Power (Tx), Spreading Factor(SF), Band Width(BW) and Coding Rate (CR). These parameters are discussed below in detail [27].

Transmission Power: T_x power is an indicator of the signal strength transmitted from the LoRa node or any other RF transmitting device. It is expressed in dBm. For most of the sensor nodes in the EU 863-870 region, the maximum transmit power for nodes is set at 14dBm which corresponds to 25 mW. Conversion of T_x power from mW to dBm can be done by using the following formula (dBm uses a fixed reference value of 1 mW):

$$T_x = 10 \log_{10} \left(\frac{P}{1mW} \right) \quad (2.1)$$

For example: A T_x power of $P = 27$ dBm corresponds to 500 mW and a R_x power of -138 dBm corresponds to $16 \times 10e - 18$ W. Transmission power of a transmitting node directly effects the power consumption and battery life. Greater the T_x power, lower the battery life will be for the transmitting device.

Spreading Factor: SF roughly means the number of data bits modulated per second. SF defines the number of raw data bits that can be encoded by the symbol. Each symbol can hold 2^{SF} chips (or steps). LoRa uses spreading Factors ranging from SF7 to SF12. SF is a very important design parameter in LoRa network. Using higher spreading factors like SF12 allows the longer communication range but decreases the data rate (slower communication).

Using higher SF increases the Time on Air (ToA) of the LoRa data packets and it also means that end node consumes more energy. Signals with different spreading factors can be transmitted at the same frequency channel without interfering with each other. For example two different sensor nodes can transmit simultaneously at frequency (say channel 868.3 MHz) with different spreading factors (say one at SF9 and other at SF12) without interfering with each other. This makes LoRa signals robust to in band interference.

Band Width: BW is the fixed frequency range in which LoRa signals are transmitted. The most commonly used bandwidths in LoRa modulation are 125 KHz, 250 KHz and 500KHz, they are usually denoted by BW125, BW250 and BW500. These common LoRa bandwidths are offered by most of the LoRa chips like SX1272 and SX1276. Band Width effects the power consumption and range significantly. Use of higher BW allows greater data rates but also consumes more power. Use of greater BW will desensitize the receiver to additional noise in channel.

Data Rate: DR in LoRa communication is derived from various combinations of SF and BW. LoRa data rates ranges from DR0 to DR15 and they are directly implemented in the end nodes. For example, DR0 in EU is combination of SF12 and BW125 (125 KHz); similarly DR6 is a combination of SF7 and BW250. Data rate effects the device power consumption, bit rate (bits/sec) and maximum payload size. More about data rates and its effects on the important parameters is discussed in the next chapter. Table 3.2 shows the allowable data rates in EU868 region. Note that all possible data rates in LoRa are not available for use.

Coding Rate: Coding rate refers to the proportion of bits transmitted that actually carries information (sensor data payload). It quantifies the degree of redundancy implemented by the Forward Error Correction(FEC). LoRa allows the following values of Coding Rates: $CR = \{\frac{4}{5}, \frac{4}{6}, \frac{4}{7}, \frac{4}{8}\}$. Coding rate can be calculated by the following equation:

$$CodingRate = \frac{4}{4 + CR} \quad (2.2)$$

In the equation 2.2 , the value of CR varies from 1-4. A higher value of CR means there are more error correction bits. But this higher value of CR also means more bits are transmitted and hence lower battery life for the end node.

2.5 Design Equations for LoRa

This section contains the important design equations and calculation examples for LoRa modulation and LoRa network design parameters. These equations are derived from Semtech Corporation's official application notes on LoRa modulation. For details see [22] [27] .

1. Basic LoRa Modulation Parameters

Symbol duration: T_s is related to SF and BW with the following equation:

$$T_s = \frac{2^{SF}}{BW} \quad (2.3)$$

Here, SF varies from 7 to 12 and BW is usually one of the following values: 125, 250, and 500 kHz.

Symbol Rate: Symbol rate is the reciprocal of Symbol duration.

$$R_s = \frac{1}{T_s} = \frac{BW}{2^{SF}} \quad (2.4)$$

$$R_s = \frac{R_c}{2^{SF}} \quad (2.5)$$

Chirp and Chips in LoRa: A chirp in LoRa is simply a ramp on spectrogram from low frequency to the high frequency. A chirp is also called as sweep signal. In LoRa modulation each chirp is divided into 2^{SF} chips or steps. For example an SF = 7 means each symbol can carry 7 raw bits and there are 2^7 i.e. 128 unique chip values or steps in this chirp.

Chip Rate: R_c is same as Band Width in LoRa. For example a BW of 250 kHz in LoRa modulation means a chip rate of 250000 chips per second. It can be calculated by the following equation:

$$R_c = 2^{SF} R_s = BW \quad (2.6)$$

Bit Rate: Bit rate while using the LoRa modulation can be calculated by the equation below:

$$R_b = \frac{SF \times BW}{2^{SF}} \cdot \frac{4}{4 + CR} \quad (2.7)$$

Where BW is Band Width, SF is spreading factor, R_s is symbol rate, T_s is symbol duration, R_c is chip rate, CR is code rate and R_b is the bit rate. Some of the wireless communication terminology used in this section for describing the design equations are discussed in the Appendix A.

2. Time on air calculation for LoRa data packets

When a signal is sent from transmitter, it takes a certain amount of time before the receiver receives the signal. ToA is the amount of time a data packet is in air and is equal to the time it takes to send message from sender to receiver. Time on Air in LoRa transmission is a very important parameter. ToA is dependent on other LoRa design parameters like SF, BW, Payload size and CR. ToA is strictly regulated by ETSI regulations because LoRa uses unlicensed ISM bands. Packet air time or ToA for LoRa data packets can be calculated by following equations:

$$T_{packet} = T_{preamble} + T_{payload} \quad (2.8)$$

$$T_{preamble} = (n_{preamble} + 4.25) \times T_s \quad (2.9)$$

$$T_{payload} = \left(8 + \max\left(\text{ceil}\left(\frac{8PL - 4SF + 28 + 16CRC - 20H}{4(SF - 2DE)}\right), 0\right) \right) \times T_s \quad (2.10)$$

T_s is symbol duration in ms

PL is Payload size in bytes

SF is spreading factor which ranges from 7 to 12

BW is bandwidth which can be 125 KHz, 250 KHz or 500 KHz

CRC is Cyclic Redundancy Check used for error detection, it can be either enabled (CRC = 1) or disabled (CRC = 0), By default it is enabled.

H is Header, it can be implicit or explicit. $H = 0$ indicates header is enabled/ explicit mode and $H = 1$ indicates disabled/implicit mode.

DE is Low Data Rate Optimize. It can be enabled (DE = 1) or disabled (DE = 0)

CR is Coding Rate (CR can be in the range from 1 to 4), By default it is 1

$T_{preamble}$ is Preamble duration.

$n_{preamble}$ is Number of programmed symbols in preamble.

$T_{payload}$ is Payload and header duration.

3. ETSI duty cycle and transmit time interval calculations

European Telecommunications Standards Institute (ETSI) is responsible for the fair usage of EM spectrum for unlicensed frequency bands. ETSI has put certain limitation on the use spectrum for the uplink air time of each device in the form of duty cycle and transmit power of the devices.

Duty Cycle: Duty cycle is the proportion of time during which a node is operated. It varies from 0.1 percent to 10 percent for uplink/downlink messages on the allowed frequency channels. Duty cycle determines the maximum total uplink ToA in a day. For example 1 percent duty cycle gives the maximum air time of 864 seconds per day.

$$ToA_{max} = \frac{1}{100} \times 60 \times 60 \times 24 \quad (2.11)$$

This maximum time on air comes out to be $ToA_{max} = 864$ seconds for 1 percent duty cycle. This ToA_{max} changes for various duty cycles allowed in the region. To calculate the duty cycle of a node, with the known packet air time T_{packet} and the number of messages $N_{message}$ sent in a day, following equation can be used:

$$DC = \frac{N_{message}}{ToA_{max}} \times T_{packet} \quad (2.12)$$

Transmit time interval: Transmit time interval governs how frequently a device can send uplink messages. Packet air time (ToA) and Duty cycle decides how long a device has to wait before sending another message.

$$T_{interval} = \frac{T_{packet}}{DC} - T_{packet} \quad (2.13)$$

Example: Consider a LoRa node which sends a data packet of size 20 bytes with data rate SF7BW125 in EU 868 region transmitted at 868.1 MHz. For this packet, allowed duty cycle is 1 percent. The calculated packet air time (ToA) T_{packet} comes out to be 72 ms. So the transmit time interval can be calculated as:

$$T_{interval} = \frac{0.072}{0.01} - 0.072 = 7.12 \quad (2.14)$$

The transmit time interval comes out to be $T_{interval} = 7.12$ seconds. So this device has to wait a minimum of 7.12 seconds before it can send another message (provided device is configured to sample at equidistant time interval).

4. Link Budget calculation for LoRa

Receiver Sensitivity: R_x sensitivity quantifies the minimum amount of signal strength (RSSI) required at which a receiver can successfully demodulate the signal in wireless communication. For LoRa it can be calculated by the following equation:

$$R_x Sensitivity = -174 + 10 \log_{10} BW + NF + SNR \quad (2.15)$$

BW is receiver bandwidth in Hz,

NF is receiver noise figure fixed for hardware implementation,

SNR is signal to noise ratio required by modulation scheme.

Noise Floor (in dBm) can be calculated by the following equation:

$$NF = 10 \log_{10}(K \times T \times BW \times 1000) \quad (2.16)$$

K is Boltzmann' constant

T is room temperature in(293 Kelvin)

Example: LoRa transceivers have a receiver sensitivity of about -137dBm for typical data rate value of SF12BW125 at room temperature. Note that this LoRa signal can be demodulated at 20dB below noise floor can be decoded successfully!

Free Space Loss: The signal between transmitter and receiver loses power as the distance gets larger between transceivers. This is called free space loss. As a rule of thumb, doubling the distance results into 6dB loss. FSPL (in dB) can be calculated by the equation below:

$$FSPL = 20 \log_{10}(d) + 20 \log_{10} Freq - 147.5 \quad (2.17)$$

d is the distance between transceivers

$Freq$ is the frequency of transmission

Link budget: It quantifies the link performance in a wireless communication system. It is the sum of all the gains and losses from the transmitter, through the medium and to the receiver. It is usually specified in the unit dBm instead of mW. It can be calculated by the equation below:

$$P_{Rx} = P_{Tx} + G_{System} - L_{System} - L_{Channel} - M \quad (2.18)$$

P_{Rx} is the expected power incident at the receiver in dBm

P_{Tx} is the transmitted power in dBm

G_{System} is the system gain (eg: from antennas) in dB

L_{System} is the system losses (eg: from wires) in dB

$L_{Channel}$ is the channel losses in dB

M is the fading margin in dB

Example: For typical values of T_x power of 14dBm, SF12BW125, and receiver sensitivity of -137dBm at room temperature, LoRa gives a link budget of 150 dB.

2.6 LoRa Devices

The main devices used to create the LoRaWAN-based sensor networks are Sensor Nodes, gateways/base stations, and servers. These devices are discussed in the following sections:

2.6.1 Sensor Nodes

LoRa sensor nodes are devices which collect data from the environment and transmit this collected data to the gateway. These devices are located at the edge of the network. These sensor nodes are also known as end devices, end nodes, sensor nodes or sometimes just as devices. These names are used interchangeably throughout this report. Sensor nodes are usually powered by small batteries. Figure 2.3 shows several LoRa sensor nodes used during prototyping while creating the wireless sensor networks.

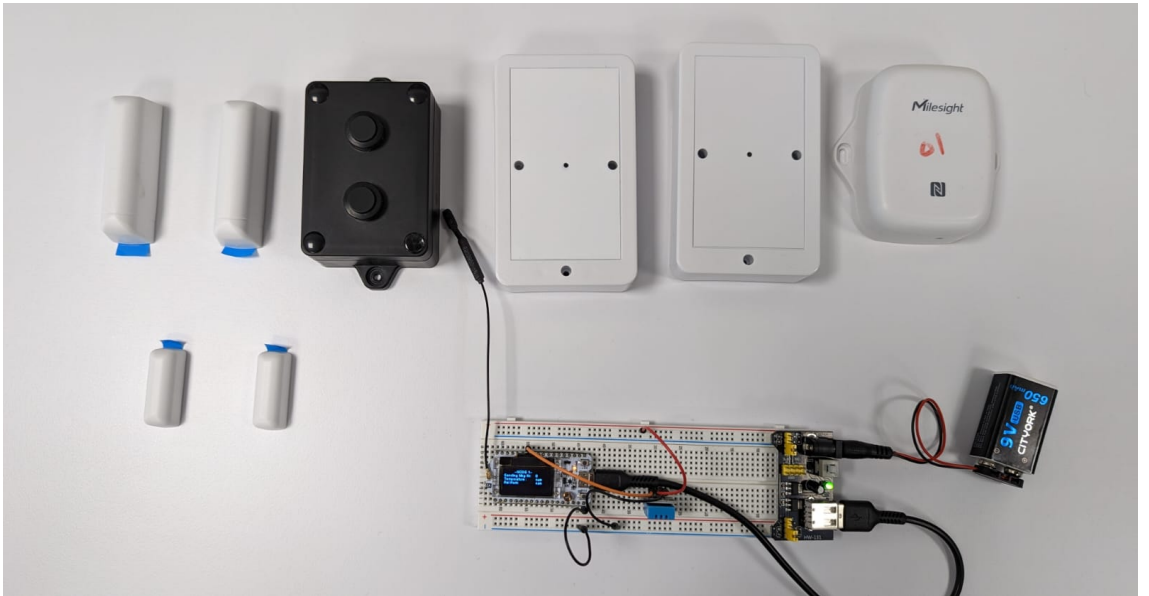


Figure 2.3: LoRa Sensor Nodes

On top right corner (marked with 01) is a commercial temperature and relative humidity sensor node. The black colored node in the middle is an ultrasonic distance sensor with accelerometer for tilt detection. Both these nodes are enclosed in IP67 enclosures for use in outdoor harsh environment. These type of sensor nodes have applications in several areas like climate monitoring in pharmaceutical and food processing industry, fluid level sensing in tanks and water level monitoring in river flood control. Between these two nodes, the white rectangular nodes are people occupancy counter sensor. In the bottom right corner of the picture, there is a LoRa development board (Heltec LoRa32 development board), a common temperature and humidity sensor module (DHT-11, blue colored), a power supply unit connected with 9V battery. All these components are connected together using jumper wires and solder-less breadboard to create a functional LoRa sensor node for rapid prototyping. These development boards along with different sensor modules are commonly used for testing new sensors, system integration and other rapid prototyping tasks. Sensor nodes made from development boards are not suitable for commercial use in outdoor conditions.

Internal architecture of sensor nodes

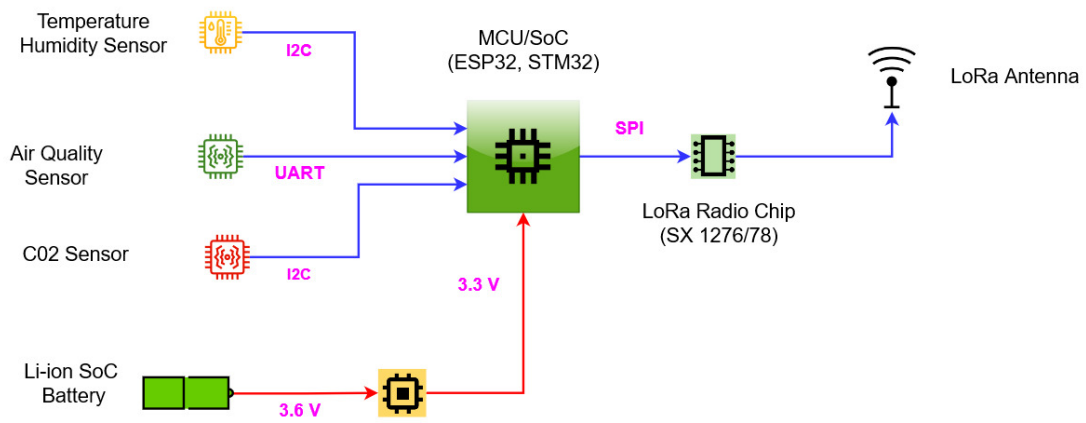


Figure 2.4: Schematic showing internal architecture of LoRa sensor node

Figure 2.4 shows an abstraction of the internal architecture of LoRa end devices. Any LoRa sensor nodes generally have the following main components:

- Micro-controller (MCU) or System on Chip (SoC) for computing, data processing, and internal communication between various modules.
- A radio module capable of LoRa communication.
- A LoRa antenna (could be PCB, 3D or external)
- One or sensors modules like temperature, humidity, pressure, ultrasonic distance, LDR, etc
- Battery for power supply (Usually AA-sized or Lion SoC batteries)
- Voltage regulator circuit

The LoRa communication chip is usually connected to SoC as an external module via serial communication protocols like Serial Peripheral Interface (SPI). LoRa radio module

can also be embedded in Soc. For example in ST microelectronics' **STM32WLE5xx** series chips, the LoRa radio module is embedded inside the SoC and does not come as a stand-alone module. Sensors like ultrasonic distance sensors, CO2 sensors, and temperature and humidity sensors are also connected to SoC via serial communication protocols like I2C, UART, and SPI as external modules. Note that, some nodes can have multiple sensors connected to a micro-controller/SoC.

2.6.2 Gateways

LoRa gateways are the devices that receive LoRa-modulated RF signals from end nodes and forward this incoming data to the application/back-end servers. Gateways are more powerful and intelligent devices that can communicate with thousands of sensor nodes over a large geographical area. They work as a bridge between sensor nodes and the backhaul internet. Gateways need to be connected to the back-haul Internet via Ethernet, WiFi, or cellular network services like LTE. The primary function of the gateway is to work as a packet forwarder to the back-end/application server. This basic functionality of packet forwarding has to be supported by every LoRaWAN gateway. Although nowadays, most of the commercial Lora gateways have inbuilt LoRa network server and application server. LoRa Network server is where the LoraWAN protocol is implemented. These embedded network servers can forward data packets to the external back-end/application servers via protocols like HTTP or MQTT. See figure 2.5 for the internal architecture of the LoRa Gateway. These are the main components of any LoRaWAN compatible gateway:

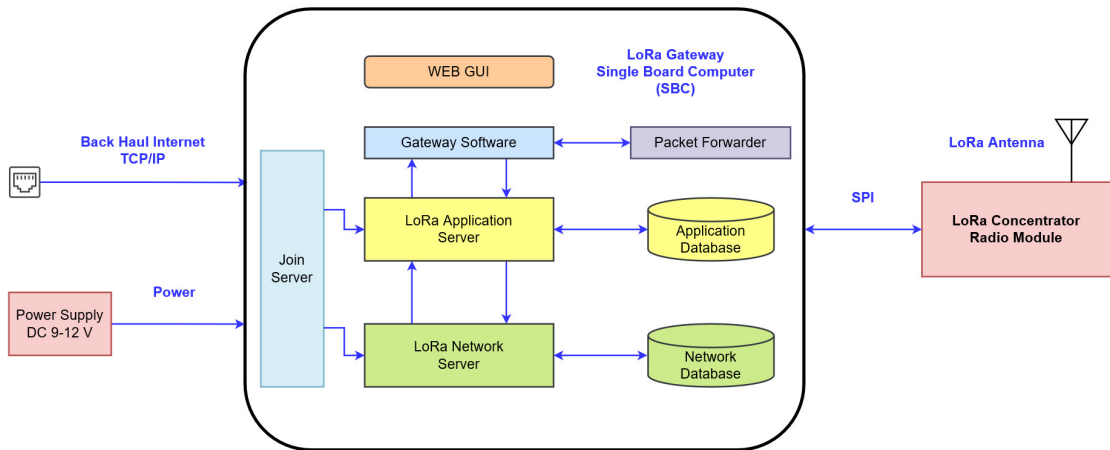


Figure 2.5: LoRaWAN gateway internal architecture

- Microprocessor or Single Board Computer (Arm based multi-core processors)
- Chips capable of connecting to backhaul internet (usually WiFi, but cellular connectivity is optional)
- Multi-channel LoRa concentrator radio module/chip. This module/chip is different from modules used in end nodes.
- Power is supply generally by DC adapter or using Power over Ethernet (PoE).
- Large inbuilt flash memory and RAM.
- Separate Antennas for LoRa, WiFi and GPS

Gateways can be accessed by Web based Graphical User Interface (WebGUI) with login credentials. All the important functions like creating applications for sensors, configuring LoRa network settings and integration with external servers can be done by using Web GUI. It is also possible to perform functions like device management, redirecting message flows, message payload decoding and manipulation via WebGUI based application in some gateways. Gateways also support functions like over the air firmware upgrades for end nodes. It is also possible to change the polling frequency of specific sensor via downlink message by gateways. Figure 2.6 shows WebGUI of a LoRaWAN gateway (RAK Edge indoor gateway) used during prototyping. GUI shows some test applications created for various sensors. Each application can have multiple sensors associated with it.

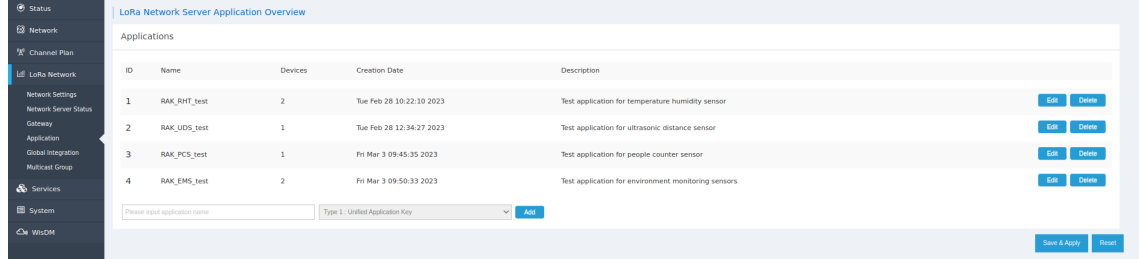


Figure 2.6: Web GUI of LoRaWAN gateway

Compared to end nodes, gateways are capable of receiving data at multiple frequencies (channel) simultaneously at various spreading factors (SF) and Band Width (BW). LoraWAN-compatible Gateways can support 8, 16, or 64 channels. 8 channel gateways are common for indoor use while 64-channel gateways are expensive and used for commercial purposes. Note that single-channel gateways are not compatible with the LoRaWAN protocol. Gateways need a constant power supply and do not run on batteries. For power supply, gateways can be connected to the mains by DC voltage adapters (9-24 V) or power over Ethernet (PoE). Gateways have more powerful external antennas to boost the range for receiving messages. Note that the LoRa concentrator modules used in the gateway are quite different from the LoRa communication chip used in sensor nodes. For example, gateways from RAK Wireless use SX1302 series chips from Semtech Corporation.

2.6.3 Servers

Network servers are one of the most intelligent unit in the entire LoRa/LoRaWAN based networks. There are various servers used in the LoraWAN network at different stages. Some of the commonly used servers in LoraWAN networks are: the Lora Network server, Lora Join server and LoRa application server. For smart gateways, small servers are embedded in the gateway itself that perform the functions for Network Server, Join server and the Application server. Note that these servers are different from the external back-end servers and "application servers" used in building end to end systems. These external servers are usually hosted on cloud or as on-premise local servers.

Servers in LoRaWAN based WSN are responsible for important functions like:

- Processing join and accept requests of the end devices.
- Receiving, acknowledging and authentication of the messages.
- Device management in the network via applications.
- Removing the duplicate messages received by the gateway.

- Implementing the network design parameters.
- Performing the Automatic Data Rate adjustment (ADR protocol).
- Security and encryption of the messages sent between sensor and the gateway
- Establishing IP connection between back end servers and the gateway via HTTP or MQTT

3.1 Introduction to LoRaWAN

LoRa PHY layer is responsible for the transmission of data between nodes and gateway over the air by the physical modulation of radio waves. LoRaWAN network protocol is a Media Access Control (MAC) layer built on top of the LoRa PHY layer, see figure 2.1. It is an open protocol defined by the LoRa Alliance. This protocol provides many features which were not present in the LoRa PHY layer. The LoRaWAN MAC layer provides features for secure and reliable communication. In the LoRa PHY layer communication was not encrypted but the LoRa MAC layer has 2-layer AES encryption for security. The protocol allows devices to send messages and get back an acknowledgment as well. This was not possible in the PHY layer. LoRaWAN is an International Telecommunication Union standard (ITU-T Y.4480). The set of rules in the LoRaWAN protocol is managed by the LoRa Alliance. LoRa Alliance is also responsible for the development of LoRaWAN open standard and promoting large-scale deployment of LPWAN IoT [3].

3.2 LoRaWAN Network

LoRaWAN protocol allows the creation of network topology. LoRaWAN network architecture follows a star-of-star topology as shown in figure 3.1. It is relatively simple to implement. This network topology is suitable for power-constrained end nodes in the network and minimizes the amount of network traffic. The three most important components of a LoRaWAN network are end nodes, gateways, and servers. LoRaWAN end devices and gateways can do bi-directional communication. In uplink communication, the node sends sensor data in form of small data packets to the gateway. While in downlink communication, the gateway sends data to the end nodes. The uplink communication is typically used for data transmission, whereas the downlink is used for control purposes. Downlink communication also allows the gateway to send data to specific nodes for firmware updates or an actuation command. Communication in LoRaWAN is heavily dominated by uplink messages.

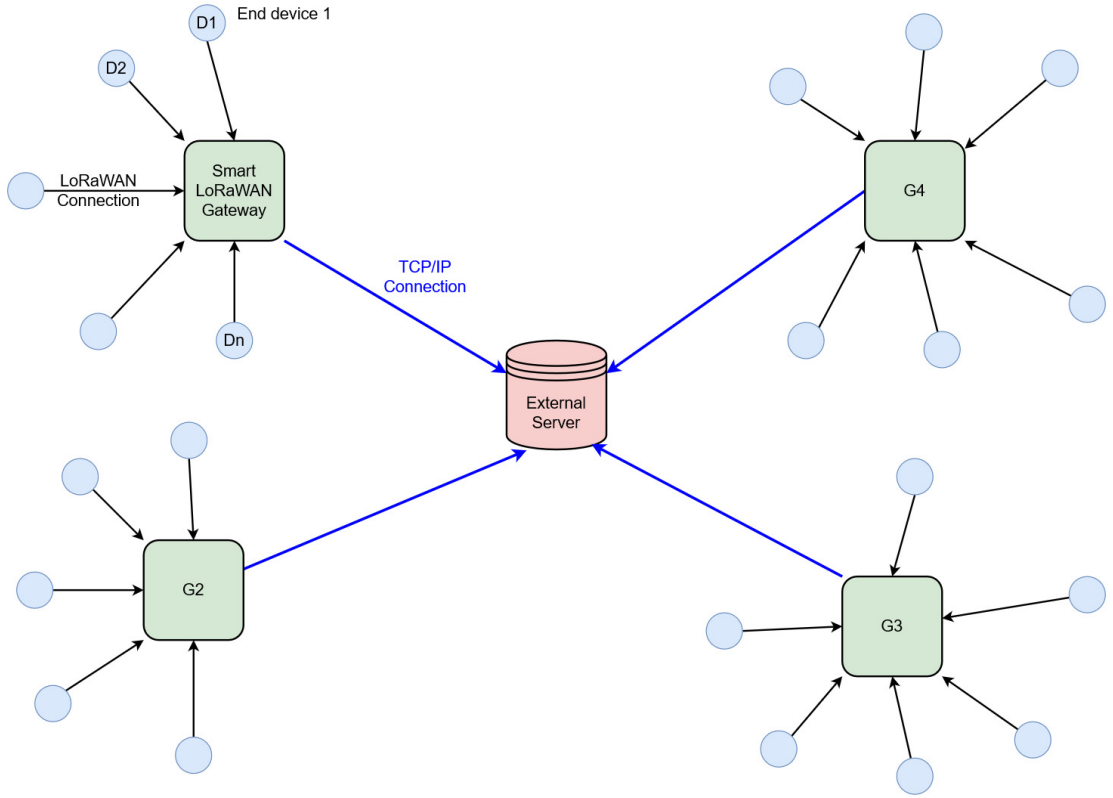


Figure 3.1: LoRaWAN network topology

Nodes are capable of up-link communication to multiple gateways and broadcast messages to all the gateways in their range. So when a node transmits the data packet, it can be received by several gateways resulting in duplicate data packets. Gateways forward the received packets to the LoRa Network Server (LNS). Network Server collects messages from all the gateways and filters out the duplicate messages and then forwards them to the LoRa application server. LNS is usually co-located in the gateway together with the application server. LNS could be a private server or a cloud service provider could be used (e.g The Things Network). Also, it is possible to host LNS externally, separate from the gateway. In this case, the gateway works just as a packet forwarder. Chirp stack is an example of one such open-source LoRaWAN network server. For visualization of information flow in the LoRaWAN network see the figure 3.2.

Co-located servers in gateway are usually resource constrained and have limited memory and processing power. Gateways need to send the incoming data to external back end servers for historical data storage and analytics. These external servers are connected to the gateway by the TCP/IP connection. Data transfer between gateway and external servers can be done by the standard communication protocols like HTTP and MQTT. Note that LoRaWAN communication is only present between end devices and gateways. [15].

3.2.1 Information flow and security in LoRaWAN

Security and encryption: In a normal LoRa radio link (PHY Layer), the data packets that are sent between the gateway and end devices are not encrypted. LoRaWAN provides 2-layer 128-bit AES encryption for secured communication. It has additional features for

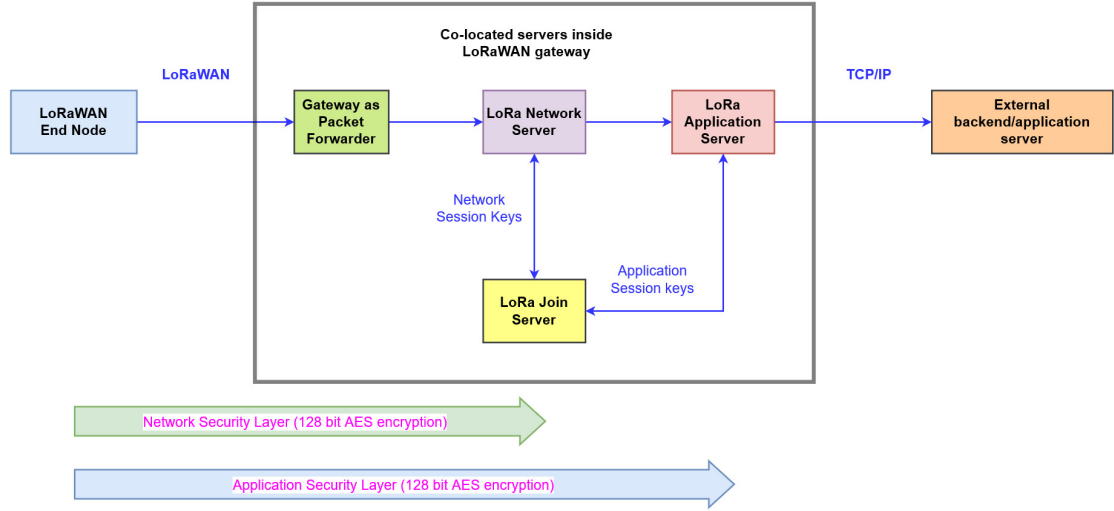


Figure 3.2: Information Flow in a LoRaWAN Network

security like the authentication of the devices while joining the network server and message integrity checks by using Message Integrity Code (MIC). Gateway checks the integrity of the incoming messages. If the message does not have the correct CRC, it will be discarded and not forwarded to LNS. LNS establishes a secure and encrypted connection between end devices and the gateway. When the gateway, LNS, and application server are not co-located, the data packets are not visible to the gateway and network server due to encryption. Only the application server can decrypt the data.

Join Procedure for end devices: Join procedure is initiated by the end device to join the network by a join request message to LNS. An authentication procedure is carried out with the gateway, LNS, join server, and application server. The join request message contains parameters like DevEUI and JoinEUI. The end device and the application server stores the unique identifiers of the device (Join EUI, application key, and DevEUI). If the application server has the same application key as the end device, then only the join request of the device is accepted and it is allowed to join the network successfully (in OTAA mode). The join server generates the network session key and application session key from the unique identifiers of the device. These keys are unique for the connection between the device and the network. Also, a dynamic address is assigned to the end device after it successfully connects to the network [25].

3.2.2 Parameters for configuring LoRaWAN devices

LoRa Alliance has defined some important parameters in LoRaWAN Specification which are required to configure the end nodes and gateways for creating a LoRaWAN network. These parameters are discussed below: [15].

- **DevEUI:** A 64-bit unique identifier for an end device in IEEE EUI64 address space. DevEUI is usually written on every device or printed as a QR code by the device manufacturer. Every manufacturer must provide the DevEUI for LoRaWAN-compatible devices.
- **Gateway EUI:** Like end-devices, each gateway also have a unique 64bit EUI. This

is known as gatewayEUI. The back-end servers can identify the gateway from which data packets are arriving by using the gateway EUI.

- **DevAddr:** It is a 32-bit dynamic address received by the device after activation. It is assigned by the network server and it is non-unique.
- **AppEUI (JoinEUI)** - This is a 64 bit unique identifier for each device in IEEE EUI64 address space. It is used for join procedure and deriving session keys. AppEUI is also known as Join EUI. It is only used for OTAA devices.
- **Application Key (AppKey):** A device specific encryption key used during Over The Air Activation (OTAA). It is used to derive the Application Session Key (AppSKey) and the Network Session Key (NwkSKey).
- **Network Key(NwkKey):** It is one of root keys like application key. LoRaWAN Version 1.0x. had only AppKey but later LoRaWAN Versions from 1.1x uses both AppKey and NwkKey.
- **Application Session Key (AppSKey):** It is 128 bit key that secures the data traffic between end device and the LoRaWAN application server. It is derived from the application key.
- **Network Session Key (NwkSKey):** It is 128 bit key that secures the data traffic between end devices and the LNS. This key is also derived from the application key after activation.

Some other important parameters which are not directly used in configuring LoRaWAN end devices and gateways include: DevNonce, Forwarding Network session integrity key (FNwkSIntKey), Serving Network session integrity key (SNwkSIntKey), Frame counters, FPort, Message Integrity Code (MIC). These parameters are not discussed here and more details about these can be found in LoRaWAN Specification [15].

3.3 Device activation in LoRaWAN

There are two different ways for end-device activation in LoRaWAN: Over the air activation and activation by personalization. The main features of these methods are discussed below:

3.3.1 Activation by Personalization

In ABP, end devices are directly tied to the network because a join procedure is passed. In ABP DevAddr and the session keys are directly stored on the end device instead of being derived from the join procedure using DevEUI, JoinEUI, and AppKey. ABP is simplified but less secure.

3.3.2 Over The Air Activation

In OTAA, end devices join the network by following a join procedure. This join procedure is used for device authentication and exchanging the cryptographic keys. It is discussed already in the previous sections. LoRa alliance recommends the use of OTAA over ABP. OTAA is considered a more secure approach for device activation. It also has the added advantage of end devices not being tied to a specific network.

3.4 Device class in LoRaWAN

LoRaWAN protocol defines the way devices should behave in communication for efficient use of battery and spectrum. Based on communication, devices have been categorized into three categories. These are discussed below [18]:

- **Class A devices:** Class A devices are generally powered by batteries and do bidirectional communication with the gateway. Class A devices mostly do uplink communication, down-link communication is used for firmware upgrades over the air or an actuation command and it is not very frequent. This communication is asymmetrical. Class A devices sleep most of the time when they are not sending data. They are highly efficient in power consumption [23].

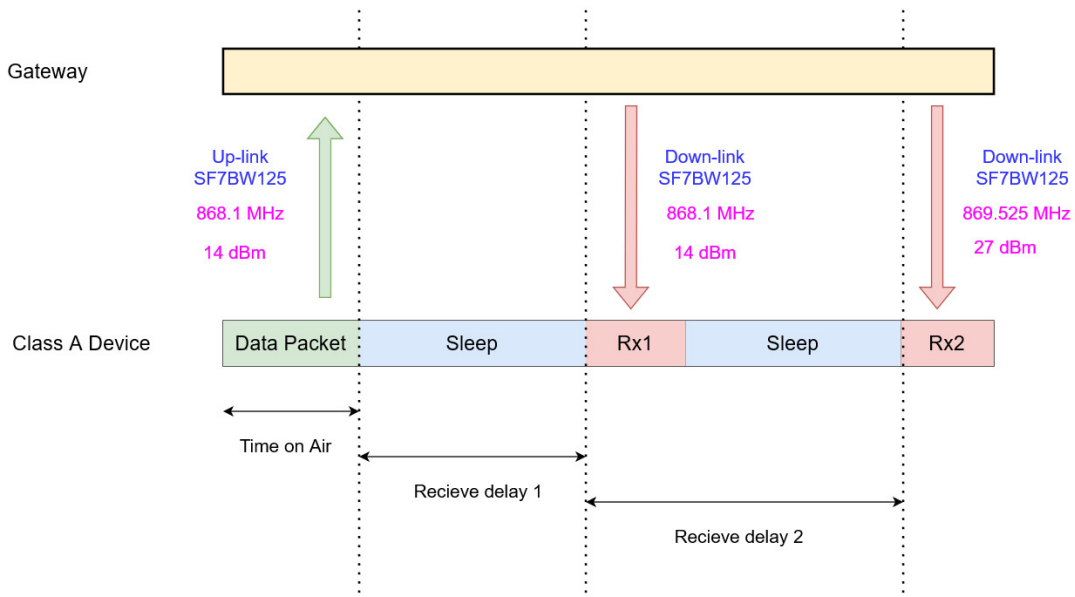


Figure 3.3: Class A device receive windows

After transmitting a data packet up-link, device opens two short down-link receive windows. Gateway can respond to any of these windows but not both. The length of receive windows must be long enough for transceiver's radio to detect downlink preamble. The length of receive window Rx1 depends on the uplink frequency and data rate. By default down-link communication in Rx1 uses the data rate and transmission frequency of last up-link message. The length of receive window Rx2 depends fixed pre-configurable data rate and frequency and does not depend on the up-link data packet's transmission frequency and data rate. For Rx2, the default down link frequency is 869.525 MHz. See figure 3.3 for better understanding of the behaviour of class A devices. Note that all LoRaWAN end devices must support class A functionality. [16].

- **Class B devices:** Class B is also known as "Beaconing" class. They are suitable for application which requires the end device to be available for downlink communication at predictable time. See figure 3.4 for the behaviour of Class B devices. Class A devices were extremely efficient at power consumption but they lacked receive windows for reliable downlink communication.

Class B devices are an improvement over Class A devices, they have extra receive windows that open at scheduled time intervals. These windows are also known as "ping slots". These devices receive a time-synchronized beacon from the gateway.

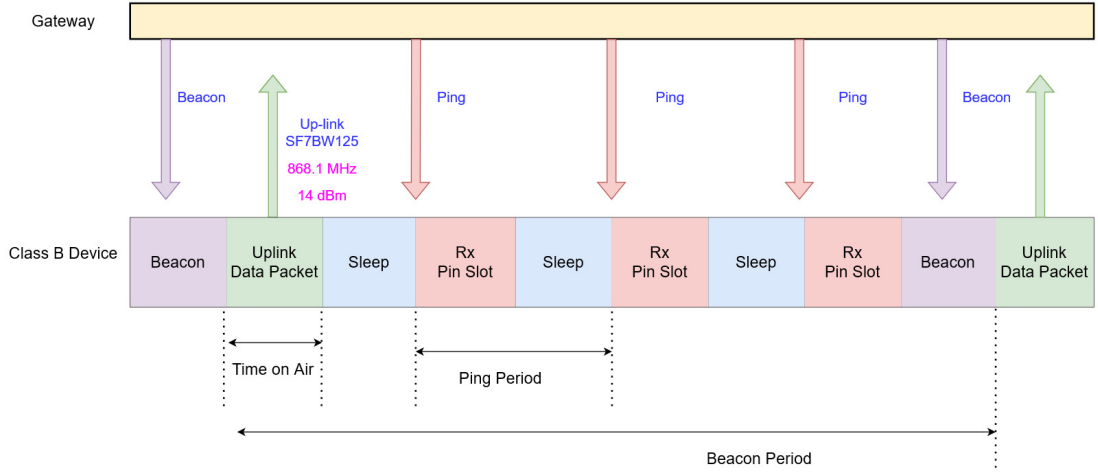


Figure 3.4: Class B device receive windows

The end device opens the receive windows (ping slot) periodically based on the time referencing of the beacon. These ping slots can be utilized by the gateway for downlink communication with the end device. Class B devices also incorporate class A behavior. Note that all devices join the network as Class A devices, later Class B functionality can be enabled by the application layer. These devices are also powered by batteries but their sleep time is controlled by the network. [24].

- **Class C devices:** Class C is also known as the "Continuously Listening" class. These devices are suitable for applications that requires the continuous down-link communication with the gateway. For example they have application in smart metering for utilities and applications which requires actuation like street lights and pumps. Unlike Class A and B devices, Class C devices are usually powered by mains not by battery. Note that Class C devices also supports class A functionality. It is possible to temporarily change the functionality of Class A/B devices to class C functionality for firmware upgrades. See figure 3.5 for better understanding of the behaviour of class C devices.

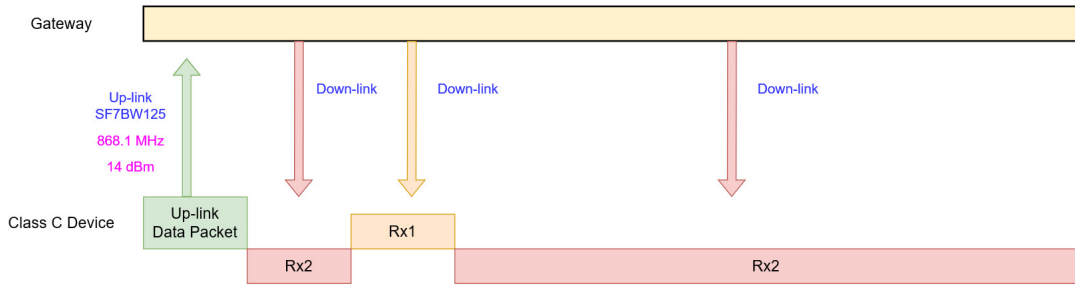


Figure 3.5: Class C device receive windows

These devices can continuously receive down-link data packets from gateway since their receive window is continuously open. Note that while doing up-link communication, Class C devices cannot receive the down-link message (half duplex behaviour).

3.5 LoRaWAN MAC Message Format

LoRaWAN Specification 1.1 defines the LoRaWAN MAC message format and the underlying data packet structure as shown in left hand side of the figure 3.6. Right hand side of the figure shows an example of how an entire up-link message (PHY Payload) actually looks like in the gateway in JSON format. This example packet was recorded during the prototyping in gateway logs. As discussed in the LoRa PHY layer, the PHY payload contains all the information about the MAC payload.

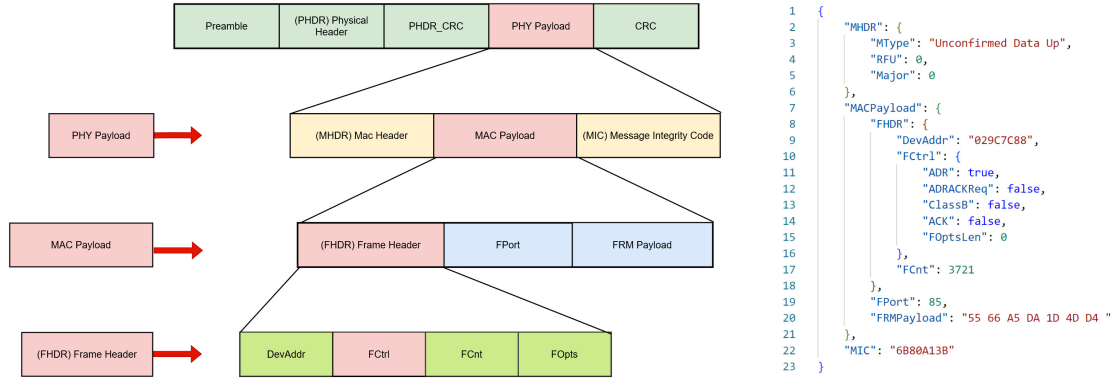


Figure 3.6: LoRaWAN MAC message format
Source: Based on LoRaWAN Specification 1.1

PHY Payload field contains the Mac Header (MHDR), MAC Payload, and Message Integrity Code (MIC). MAC header gives information about message type, the format of the message (Major), and RFU. MIC is used for preventing the tampering of messages by bad actors. The MAC Payload field contains the Frame Header (FHDR), FPort, and FRM Payload fields. FPort is the port number and varies for different sensors (Usually a manufacturer uses the same FPort for all of its sensors). FRM Payload is the main field containing the sensor payload. FHDR contains the end device address (DevAddr) and information about the Frame counter (FCnt), Frame control (FCtrl), and Frame Options (FOpts). FCtrl further contains the information about ADR, ADR acknowledgment request, device class, and frame options length [15].

3.6 Adaptive Data Rate

LoRaWAN defines Adaptive Data Rate (ADR) protocol to dynamically control and optimize the important up-link transmission parameters of LoRa end devices like SF and BW. ADR is controlled by the LoRa network server. It works by calculating the link budget by measuring SNR and RSSI. It then optimizes the data rate based on available link budget. ADR algorithm can be easily enabled in LoRaWAN gateway and end node. It greatly simplifies the process of selecting the right design parameters for radio link between gateway and the end device. For more details on the implementation and working of ADR see [1] [13] [17].

3.7 LoRaWAN Technical Specifications

Some very important rules of LoRaWAN Specification 1.1xx are discussed in this section. These rules must be followed in order to avoid heavy penalties by ETSI in the EU.

- **Radio bands** It is common for manufacturers to ship the LoRa radio module with both EU868 and US915 capability. Developer should ensure that only allowable ISM bands for regions are used. More detailed list of allowable ISM bands for LoRa by countries can be found in LoRaWAN regional parameters document. [16]:

In EU, ISM band frequency (863-870 MHz) use in LoRaWAN communication must comply with the following additional rules:

- **Maximum Transmission Tx power:** Maximum allowed transmission power varies for up-link and down-link communication. These limits are stated below:

Up-link : Maximum Transmission Tx power is limited to 25mW (14 dBm).

Down-link : Maximum Transmission Tx power is limited to 500mW (27 dBm).

- **Duty cycle restrictions:** ToA and Duty cycle decides how long end devices have to wait before sending messages on the air. In EU, ETSI EN300.220 standard regulates the duty cycle depending on channel. Duty cycle usually take following limitations: 0.1%, 1%, 10 % For example in channel (868.0 – 868.6 MHz), the allowable duty is: 1%.
- **Maximum Antenna Gain:** The maximum allowed antenna gain for end device is +2.15 dBi.

3.7.1 LoRa Data Rates

The table 3.1 shows the relationship between LoRa data rates and its effect on bit-rate (bits/sec) and Maximum Payload Size in US 902-928 region. IN US 915 region the data rates 5-7 and 14-15 are not defined [25]

Table 3.1: Effect of Data Rates on bit-rate and maximum payload size in US

Data rate	Spreading Factor	Band Width	Bit-rate	Maximum Payload size	communication
DR0	SF10	BW125	980	11	uplink
DR1	SF9	BW125	1760	53	uplink
DR2	SF8	BW125	3125	125	uplink
DR3	SF7	BW125	5470	242	uplink
DR4	SF8	BW500	12500	242	uplink
DR8	SF12	BW500	980	53	downlink
DR9	SF11	BW500	1760	129	downlink
DR10	SF10	BW500	3125	242	downlink
DR11	SF9	BW500	5470	242	downlink
DR12	SF8	BW500	12500	242	downlink

Table 3.2 shows the relationship between LoRa data rates and its effect on bit-rate (bit-s/sec) in EU 863-870 region. IN EU 863-870 region the data rates 8-15 are reserved for future use and not defined. Also the default data rate for uplink communication is DR0. Unlike US 902-928 region, EU 863-870 MHz region uses a different scheme of data rates for downlink messages. [14]

Table 3.2: Effect of Data Rates on bit-rate and maximum payload size in EU

Data rate	Spreading Factor	Band Width	Bit-rate	Maximum Payload size
DR0	SF12	BW125	250	59
DR1	SF11	BW125	440	59
DR2	SF10	BW125	980	59
DR3	SF9	BW125	1760	123
DR4	SF8	BW125	3125	230
DR5	SF7	BW125	5470	230
DR6	SF7	BW250	11000	230
DR7	FSK	50kHz	50000	230

3.7.2 Relationship between SF, Time on Air and Receiver Sensitivity

Table 3.3 shows the relationship between LoRa Spreading factor and its effects on Time on Air (ToA) and receiver sensitivity. Band Width is kept fixed at 250 KHz and CR at 2 [27].

Table 3.3: Effect of SF on ToA and Rx sensitivity (BW= 250, CR = 2)

SF	ToA(ms)	Sensitivity (dBm)
SF8	039.2	-124
SF10	132.1	-129
SF12	528.4	-134

Table 3.4 shows the relationship between LoRa Spreading factor and its effects on Time on Air (ToA) and receiver sensitivity. Band Width is kept fixed at 125 kHz [26].

Table 3.4: Effect of SF on ToA and Rx sensitivity (BW= 125 KHz, CR = 1)

SF	ToA(ms)	Sensitivity (dBm)
SF7	41	-123
SF8	72	-126
SF9	144	-129
SF10	288	-132
SF11	577	-134.5
SF12	991	-137

This chapter provides a theoretical framework for MQTT. The scope of this chapter is limited to the key concepts and knowledge required for working with the MQTT protocol. The chapter broadly covers a wide range of topics related to MQTT, such as its MQTT architecture, MQTT topics, messages Quality of Service levels, and the interactions between broker and clients. It covers the fundamental concepts and knowledge required to work with the MQTT protocol that was used to create the prototype WSNs in subsequent chapters. A reader with experience in working with MQTT can skip this chapter. There is a vast amount of literature available on MQTT and its hands-on implementation in various programming languages. This chapter is a very brief introduction to key concepts required in working with MQTT. For a comprehensive view of MQTT see [12] [8].

4.1 Introduction to MQTT

MQ Telemetry Transport (MQTT) is a light weight message transport protocol suitable for Machine to Machine communication (M2M) and Internet of Things (IoT). It has a publish/subscribe model for communication where clients can publish (publisher) and subscribe (subscribers) to the topics of interest on which messages were being published. In MQTT, publishing and subscribing clients are decoupled and do not directly communicate with each other. Communication happens through a central broker only. Messages are the key entity for communication between broker and clients. MQTT is an open protocol standardized by ISO (ISO/IEC 20922:2016) MQTT is present at the application layer of TCP/IP model. MQTT broker establishes a TCP connection with clients. MQTT is the application layer of TCP/IP [2] [21].

4.2 MQTT communication architecture

There are three important entities in any MQTT based system: The publishing clients, the subscribing clients and the broker. These are shown in figure 4.1.

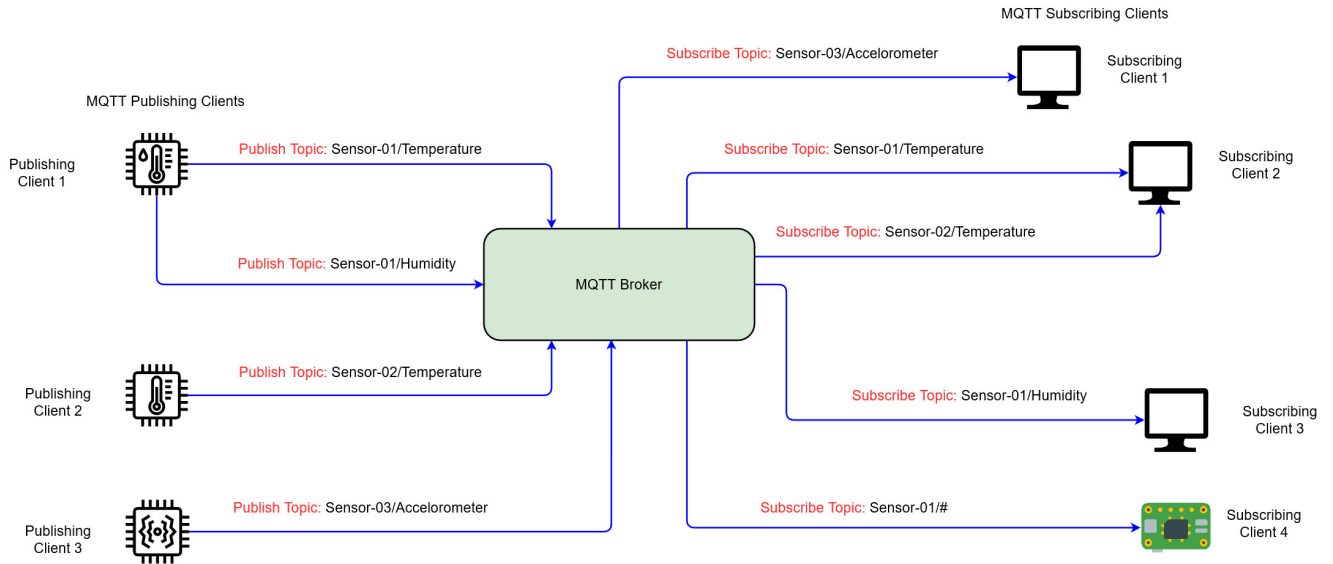


Figure 4.1: MQTT network architecture

MQTT broker: MQTT server is the machine where the MQTT broker is hosted. MQTT server can range from a simple SBC like raspberry pi to a high-end enterprise server. MQTT broker acts as the central hub of communication between publishing and subscribing clients. It is responsible for managing client connections and receiving and, filtering messages from publishing clients. It determines which clients are subscribed to which topics and then forwards the messages to the subscribing clients. Since the broker is the hub of communication so it makes the broker a single point of failure. This failure can be avoided by hosting multiple brokers as a broker cluster. In case of failure of one broker, other brokers would still be available in the cluster. This increases the reliability of the overall system. The service of broker cluster is provided by commercial MQTT vendors like HiveMQ [7]

MQTT clients: An MQTT client is any device that runs MQTT client scripts and establishes the connection with the MQTT broker over the network. MQTT clients can run on devices ranging from resource-constrained MCUs to typical PC as well as large commercial-grade servers. Both publishers and subscribers are MQTT Clients.

MQTT topics: Topics are UTF-8 strings with hierarchical structure with one or more level. Topic levels are separated by forward slash. MQTT Topics are used for filtering messages for the connected clients. Topics are case sensitive. MQTT topics also use wild card pattern with plus and hash symbols.

Working of MQTT with an abstract example:

Consider the MQTT network shown in figure 4.1. In this network 3 sensors are acting as publishing clients. These sensors are publishing sensor data as messages to the central broker on various topics as depicted in figure. Three PCs and a single board computer are acting as subscribing clients. These clients are only subscribed to the topic of their interest.

For example "Subscribing client 2" in this network is getting temperature data from both sensor-01 and sensor-02 because it is subscribed to the topics "Sensor-01/Temperature" and "Sensor-02/Temperature". Another subscribing client (client 4) is subscribed to all the data from sensor-01 by using a wildcard. In this way subscribing clients only get the data they are interested in. There is no need to store and search all the data from all the sensors, clients only stores and process the data they are interested in. The failure of

one publishing/subscribing client does not effect the other clients. The main advantage of this publish-subscribe model is that it decouples the publishers and subscribers. This Publish/Subscribe model scales much better than traditional Client-Server model.

Quality of service: MQTT offers 3 levels of Quality of service (Qos). It determines the level of guarantee for message delivery in the system. These different QoS offers a trade-off between message delivery guarantee and bandwidth [7].

- QoS 0 : Offers at most once delivery. QoS 0 is used when message loss is acceptable and message queuing is not required. These messages are never queued and no acknowledgement is sent back to the publisher. QoS is most efficient in terms of network bandwidth usage. QoS 0 is suitable for sending sensor data where some data loss is acceptable.
- QoS 1 : Offers at least once delivery. QoS 1 is the default level and guarantees delivery of the message. Messages at QoS 2 are re transmitted until an acknowledgement is received by the publisher.
- QoS 2 : Offers exactly once delivery. QoS 3 offers low performance and takes more bandwidth. QoS uses a handshake mechanism to ensure the exactly once message delivery. This is the highest level of assurance for message delivery in MQTT.

4.3 MQTT control Packets:

MQTT uses various different type of data packets (control packets) to perform specific functions in the protocol. Control packets have three main parts: Fixed header, Variable header and Payload. Some important control packets are listed in table 4.1. For a more comprehensive explanation of MQTT control packets, their structure and types see [2].

Table 4.1: MQTT Control Packets

Name	Value	Direction of Flow	Description
Reserved	0	Forbidden	Reserved
CONNECT	1	Client to Server	Request to connect to server
CONNACK	2	Server to Client	Connect acknowledgment
PUBLISH	3	Bidirectional	Publish message
PUBACK	4	Bidirectional	Publish acknowledgment

4.4 What makes MQTT suitable for IoT applications?

Some characteristics of MQTT which make it suitable for IoT applications are listed below:

- MQTT Clients have very small code footprint. MQTT client scripts can run on a wide range of devices. MQTT can work on resource-constrained embedded devices like microcontrollers, SoCs like ESP32s, raspberry-pi Pico W, Single Board Computers (SBCs) like Beagle-bone black and Raspberry Pi, etc.
- MQTT is suitable for event-driven networks. It is built for push-based communication, unlike HTTP which has a request/response model.

- MQTT is a data-centric protocol whereas protocols used for the web like HTTP/HTTPS. Data packet size is very small in MQTT compared to HTTP.
- It offers bi-directional communication between clients and broker. MQTT communication is data agnostic i.e. clients can send any data in formats like JSON, XML, or plain text strings. This allows the coexistence of IoT devices in a heterogeneous system with clients running in different programming languages like C, CPP, Python, JavaScript, etc.
- MQTT can work with unstable internet connections having limited bandwidth.
- It is suitable for building scalable systems with thousands of devices connected to the broker.

The most promising technology competing with MQTT for IoT communication is Constrained Application Protocol (CoAP). For a detailed comparison of communication protocols suitable for different IoT applications see [20].

Overview of System Architecture and Network Implementation

This chapter gives a very brief overview of the system architectures created during tests and experiments. This overview is not a substitute for the detailed network architectures and configuration of the various test setups implemented during this Thesis. The final prototype WSN created by using a cloud server is built in multiple iterations of tests and experiments. Figure 5.1 shows a very simple abstraction of the overall information flow through various network devices in the WSN. The communication protocols used between the different network devices for example LoRaWAN between the sensor node and the gateway and MQTT between the gateway and the server are also shown in the figure. Note that the actual sensor network architecture implementation is much more complex. A lot of changes were made in the network configuration to build the final prototype. These initial experiments and network implementation are discussed in the following chapters:

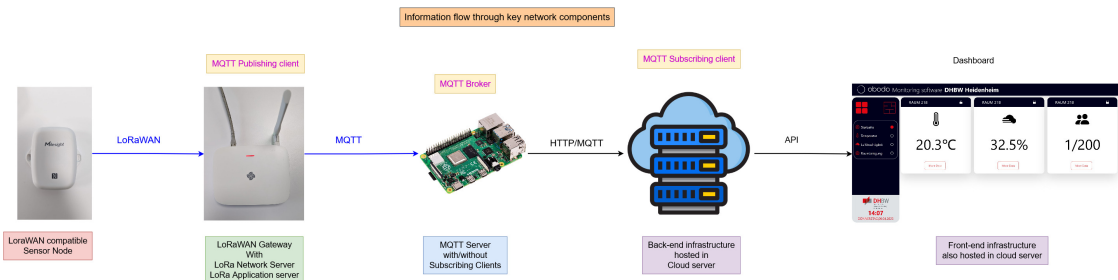


Figure 5.1: Abstraction of simplified network architecture

Chapter 6: This covers the preliminary tests with the LoRa PHY layer to establish a point to point radio link for sending sensor data. The system architecture as shown in figure 6.1 used in these early tests is very simple and only suitable for laboratory use. This chapter also explains how to configure various LoRaWAN-compatible network devices. A small section on Node-Red and its use is also provided. A very important section on decoding the sensor data and how to modify the message payload data structure is also discussed, see figure 6.9.

Chapter 7: This chapter contains WSN implementation, explaining the various sub-systems and network devices, how the data logging system was created using Node-

Red/python scripts, and the network security implementation. See figure 7.1 for the detailed network architecture. Note that only local on-premise servers were used in the implementation of this network, no cloud computing services were used. Chapter 8 discusses the results and analysis of data collected from this WSN.

Chapter 9: This chapter contains the Network architecture of the final WSN with cloud computing. This prototype WSN was based on the learning from all other previous experiments. This WSN was successfully deployed for a real-world application. Network I architecture as shown in figure 9.3 is quite similar to the deployed Network II 9.4. A short introductory section on cloud computing (Microsoft Azure) discusses the need for cloud computing compared to local on-premise servers and also briefly explains the various subsystems.

Initial Experiments with LoRa PHY and LoRaWAN MAC

This chapter covers the initial experiments done using LoRa PHY layer. It also covers the important aspects of configuring commercial LoRaWAN compatible network devices like sensor nodes and gateways.

6.1 Tests with LoRa PHY layer

A node-to-node radio communication link was created between two LoRa nodes. One development board was used as a transmitter node and another board was programmed as a receiver module. LoRa PHY layer was used for this point-to-point wireless communication.

6.1.1 Objective and description of tests:

A series of tests were conducted to check the ability of LoRa communication to send sensor data over the long range. The goal of the tests was to send temperature and relative humidity data from a LoRa sensor node (acting as a transmitter) to another LoRa receiver module. The receiver module was stationary and located inside the building for most of the tests. The position of the transmitting node was changed in most of the tests. Figure 6.1 shows the schematic of the system setup used during the tests. Heltec automation's LoRa development boards were used for transmitting as well as receiver nodes.

6.1.2 Transmitting node and receiver module

Transmitting Node: Figure 6.2 shows the transmitting node created by using the LoRa development board and other electronic components. The LoRa development board, temperature and humidity sensor module, and 3.3V/5V Power Supply Unit (PSU) were mounted on a solderless breadboard. A 9V battery was connected to PSU via a 2.1mm barrel jack connector. This power supply unit can take 9-12 V as input and is capable of supplying 3.3V as well as 5V output voltage to the breadboard. Both the sensor module

Schematic of System Setup for Node to Node LoRa Communication

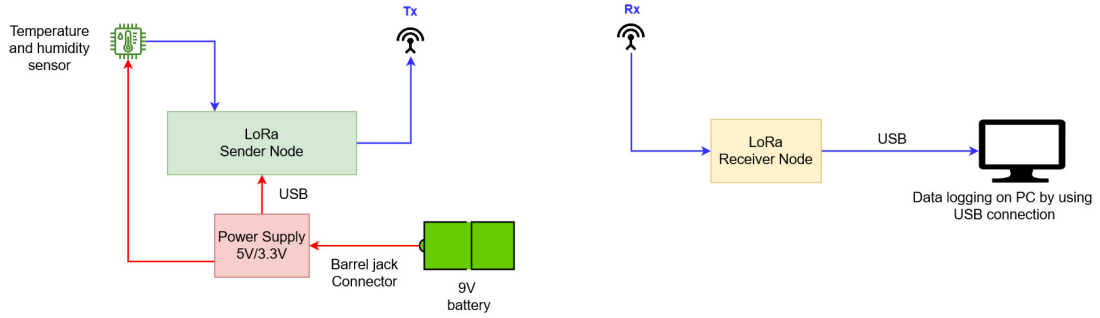


Figure 6.1: Node to node communication system setup

and development board works on 3.3V. The development board can be supplied by 5V as it has an internal voltage regulator to step down the voltage to 3.3V. The sensor module was supplied voltage through a jumper wire from a 3.3V line and the development board was supplied power via a USB cable connected to the PSU. All components were getting power from a single 9V 650mAh rechargeable Li-ion battery. Temporary connections for power, ground, and data transmission were made using jumper wires. The sensor module transmits data to the development board via a single wire. Lora development board has an inbuilt 0.96-inch Oled display which was programmed to display sensor readings and help in debugging. Hardware components used for the transmitting node are listed below:

- Heltec automation's LoRa32 development boards (with built in Oled display). Figure 6.3 shows the ESP32 SoC module and LoRa radio module (Semtech's SX1276) in the LoRa development boards used for the tests.
- DHT11 sensor module measuring temperature and humidity.
- Solderless breadboard for easily connecting electronic components.
- 3.3V/5V bread board power supply unit.
- 9V battery for power supply.
- Jumper wires, USB cables, barrel jack connector.

Receiver module: Another LoRa development board was programmed to be the receiver module as shown in figure 6.4. This receiver module was programmed to receive the incoming LoRa data packets. It displays the incoming sensor data on an inbuilt Oled display and is additionally capable of logging the data to a PC via serial communication through PuTTY. The receiver module was connected to a PC via a USB cable for data logging. This receiver node can also be connected to a power bank via a USB cable and received data can be seen on the Oled screen without any need for a PC. This was used for testing the range of the communication link.

6.1.3 Software programming of development boards

The software for both the transmitting node and the receiver module was programmed using C++ with Arduino framework. Heltec's "ESP32 develop framework" (libraries for LoRa and Oled display) were used to program the development boards. Several other

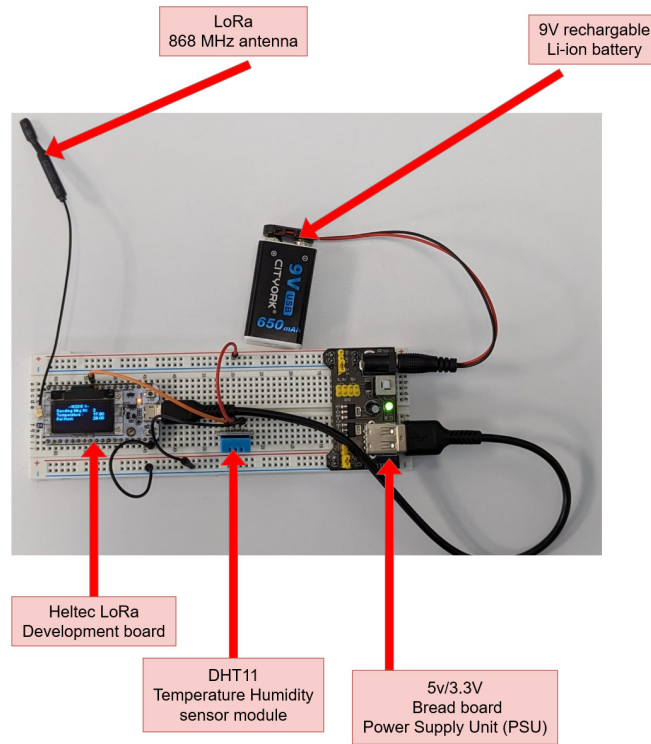


Figure 6.2: LoRa development board programmed as transmitting node

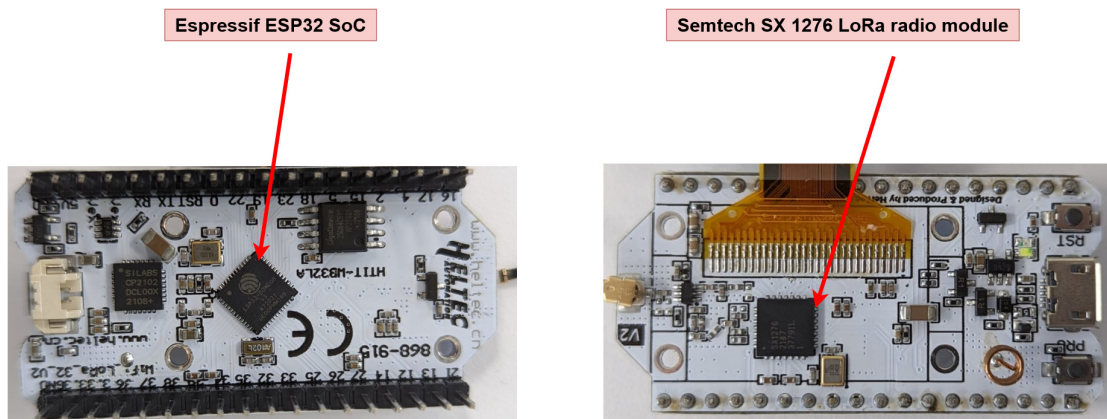


Figure 6.3: Heltec LoRa development board

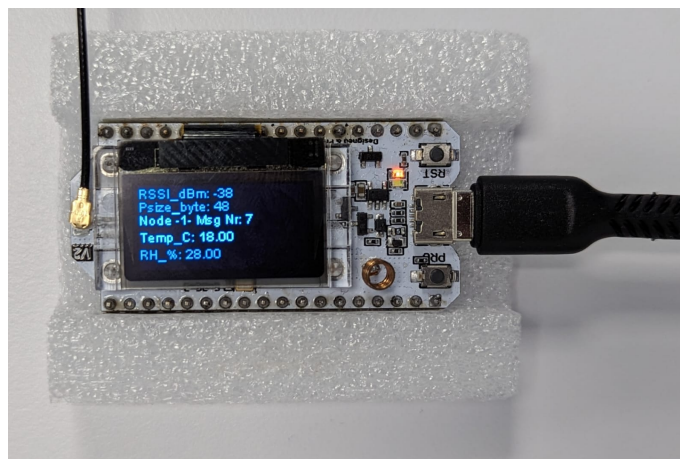


Figure 6.4: LoRa development board programmed as receiver node

C++ libraries were also used from the manufacturer of the development board and sensor modules to create the custom program. These libraries are listed below:

1. For LoRa communication: heltec.h
2. For temperature and humidity sensor: DHT.h
3. For OLED display: images.h

Message_Number	Temp_Celsius	RH_Percent
1	27.3	45
2	27.3	46
3	27.3	46
4	27.3	45
5	27.2	45
6	27.2	45
7	27.2	45
8	27.2	45
9	27.1	45
10	27.1	45

Figure 6.5: Logged sensor data from the transmitting node

The software written for the transmitting node performs several important functions. These important actions are summarized below:

- Turns "on" and initializes the Oled screen. Displays the initial message "Starting Node Nr: 1."
- Reads the sensor data and displays it locally on the Oled screen as shown in figure 6.2. Oled screen was programmed to additionally display a message number and the node number (like a device Id/name).
- Transmits a data packet containing the sensor data via LoRa PHY layer. The LoRa transmission frequency and the Tx power for all the transmitted packet was fixed at 868 MHz and 14 dBm respectively. The sensors were polled every 30 seconds. This transmitted data packet also contained the message number. The message number is automatically incremented after sending the message.
- The programmed software also had the ability to log the transmitted data packets in CSV format if the transmitting node is connected to a PC. This was done by using a USB serial connection via PuTTY. Figure 6.5 shows the logged sensor data from the transmitting node (only the first 10 entries in the CSV file).

The program for the receiver module performs the following actions:

- Displays the initial message "waiting for message" when power is plugged in.

- Receives the incoming LoRa data packets and displays them on Oled screen. It blinks an onboard LED after receiving the message.
- Receiver module also displays the RSSI value and the packet size (in bytes) in addition to the contents of received data packet as shown in figure 6.4.
- On connecting the receiver module with PC, the program could log the incoming data packets in plain text file by using PuTTY through a USB serial connection. Logged data packets from the receiver module are shown in figure 6.6

```

RSSI_dBm: -75
Psize_byte: 48
Node -1- Msg Nr: 1
Temp_C: 23.60
RH_%: 52.00

RSSI_dBm: -73
Psize_byte: 48
Node -1- Msg Nr: 2
Temp_C: 23.70
RH_%: 52.00

RSSI_dBm: -67
Psize_byte: 48
Node -1- Msg Nr: 3
Temp_C: 23.50
RH_%: 52.00

```

Figure 6.6: Logged sensor data from the receiver module

6.1.4 Observation and results of tests

Important parameters for the test are listed below:

Important parameters for the test are listed below:

- Node transmission power: 14dBm
- LoRa Frequency: 868 MHz
- Time for initialization of SoC, Oled display, and DHT11 sensor: 5 seconds
- Sampling (by polling, no hardware interrupts used): 30 seconds (29.5 + 0.5)
- Power supply to the sensor module: 3.3V by 9V battery connected to the power supply unit.
- Power supply to LoRa SoC: 3.3V by same 9V battery connected through USB cable
- Transmitted data packet size: 44-50 bytes

- Baudrate for serial communication: 115200 (for receiver module connected to PC)

Range and Power consumption

For point to point communication, the range between LoRa transmitting node and the receiver module depends on the factors like environment (Urban or rural), transmission power (Tx), Fresnel zone radius, Link margin, frequency of LoRa transmission, antenna gain and the sensor sampling period. Indoor coverage further depends on type of building material and obstacles. For the tests conducted with PHY layer, receiver module was able to get the messages at a distance close to 500m with RSSI: -130 dBm. Beyond that range packet drop rate was quite high. Receiver module was only able to get 1 out of every 2-3 data packets transmitted. Since in these preliminary tests the transmitting node has the Oled screen in the "On" position for entire duration of test run, the battery performance was quite poor. The node was able to a maximum of 867 messages (at 2 messages per minute) before the battery ran out. Note that battery optimization was not the goal in these initial tests.

6.1.5 Limitations of the tests with Lora PHY

- Packet size: In the PHY layer communication, data packet size was varying between 41-50 bytes, which is too large for just transmitting temperature and humidity. This large packet size was due to the reason because ASCII character which are not part of actual sensor data were being transmitted.
- Security: No encryption was implemented for the security between the transmitting node and receiver module. Anyone who is not the intended recipient of messages can capture the data packets with a programmed LoRa receiver node.
- Power Consumption: The power consumption of the sender was too high. A 650 mAh 9V battery was able to send messages only for 8-10 hours before running out. Oled display on the sender node was also consuming a lot of power. For commercial sensors, Oled screen is generally not used.
- Reliability: The packet drop rate was high with the test setup based on the PHY layer. During the tests, the receiver module will miss to capture 1-3 data packets sometimes. Sometimes the SoC on the transmitting node will fail to read the connected sensor properly. This would result into sensor reading being displayed as "nan"
- Scalability: This node to node communication setup is only suitable for lab experiments, testing and integration of new sensors for prototyping. This is not ideal for the commercial deployment.

In the tests with LoRa PHY layer there was no mechanism for important actions like device management, over the air firmware upgrade and sending downlink message to change polling frequency. The solution to these limitations is already provided by the commercial sensor nodes implementing the LoRaWAN MAC protocol. For further tests and experiments LoRaWAN compatible commercial sensor nodes were used.

6.2 Tests with LoRaWAN MAC Layer

To improve on the limitations of LoRa PHY layer, commercial LoRaWAN compatible sensor and gateway (Milesight UG65 LoRaWAN indoor gateway) was used. Figure 6.7 depicts the system abstraction and the flow of information in a simple network created by using sensor nodes and gateway compatible with LoRaWAN protocol. How these network devices were configured is discussed in the following sections:

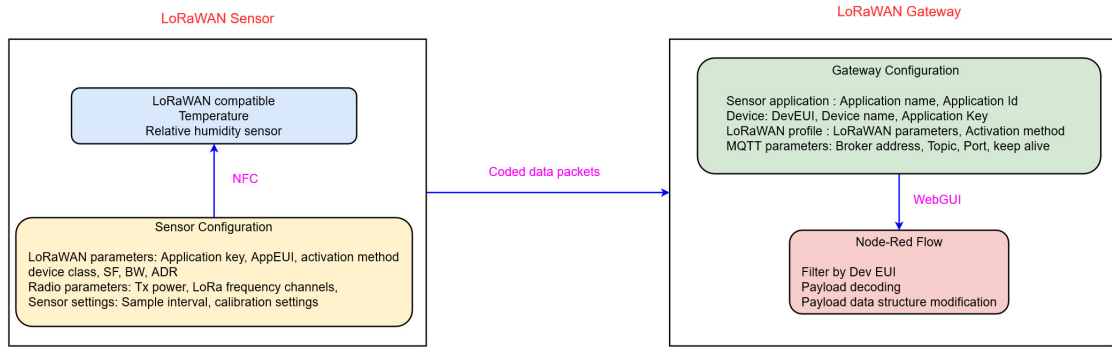


Figure 6.7: System abstraction and information flow

Configuring network devices

Configuring sensors: This section discusses in brief how the sensors (used in the prototype WSN) can be configured with an example of Milesight’s EM300 series temperature and relative humidity sensor. This sensor was configured using an inbuilt NFC reader and the mobile toolbox application from the manufacturer of the device. The older versions of sensors required a different method to configure them. They needed to be connected to a PC with a USB cable to flash programs or to make changes to the existing configuration. As shown in figure 6.7, important LoRaWAN and radio parameters like application key, AppEUI, data-rate (SF+BW), device class LoRaWAN MAC version, Tx Power, frequency channels, sensor sampling interval were configured in the mobile toolbox app. This configuration was written to the sensor by using NFC. These parameters are discussed in great detail in the chapter on the LoRaWAN MAC layer.

Configuring gateway: Configuring the LoRaWAN gateway involves several steps. These steps are summarized below for Milesight UG65 gateway with examples:

- Gateway needs to be powered and connected to the backhaul internet via Ethernet cable. Its WebGUI can be accessed by PC through the default login IP address and login credentials. Gateway could be assigned a static or a DHCP IP address in network settings. The default packet forwarder is on port 1700 by semtech corporation.
- Radio settings like LoRa frequency channels were configured in Network settings tab using WebGUI.
- The most important step was creating the LoRaWAN application and assigning the sensors to this application. The LoRaWAN parameters like DevEUI, application key, device profile etc should be same as configured in the sensor for successful connection.
- In the same application MQTT connection details can be configured for the destination of incoming LoRa data packets from the sensors.

Configuring local server: In the LoRaWAN gateway, the application requires the destination for forwarding the data packets. This is usually a back-end server. This server can be configured as local MQTT server on a dedicated Windows/Linux machine or cloud computing services like Amazon Web services (AWS) or Microsoft Azure can be used. Each one has its own advantages and limitations. For the initial tests with LoRaWAN MAC layer, an embedded network server in the Milesight's UG65 gateway was used to see the incoming data packets.

6.2.1 Node-RED:

Node-RED is a GUI-based programming tool used for connecting hardware devices, APIs, and dashboards for IoT applications. It is a web-based flow editor that runs on top of open source server environment Node.js. The key entities in Node-RED are the blocks of JavaScript code/functions called "Nodes". Data that passes through these nodes are called "messages". The message flow and payload of these messages are stored in the form of JavaScript objects and JSON files. JavaScript Object Notation (JSON) is a widely used text-based format for representing structured data. The complete message flow diagram in Node-RED is saved as a large nested JSON file that contains all the information about the flow and functions. With the help of functions written in JavaScript, the payload of these messages in flow can be modified, and desired objects can be accessed, added, or deleted from the nested JSONs. This is a very useful tool for designing the payload suitable for storing in databases or data logging. Milesight's UG65 gateway can run an instance of Node-Red. This function was used for debugging and logging the data packets during the tests.

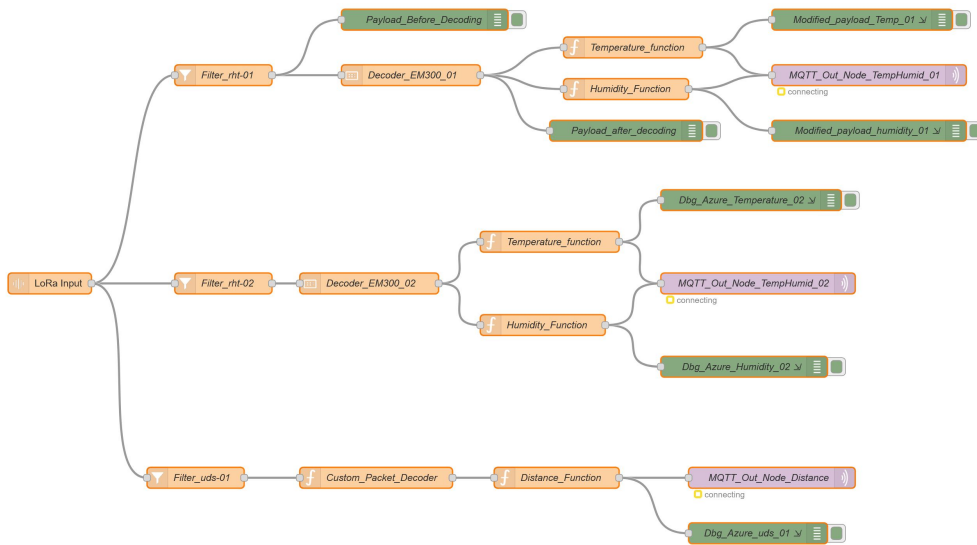


Figure 6.8: Message Flow diagram implemented in LoRaWAN gateway using Node-RED

Message flow in Node-RED: Figure 6.8 shows a message flow diagram created by using Node-RED. This flow diagram was implemented in the gateway to see the payload of incoming data packets from the sensors. With the help of this flow diagram, the received LoRaWAN messages at the gateway (Lora input node) were filtered (for specific DevEUI) and decoded using a decoder script from the sensor manufacturer. Then the message payload coming out of the decoder nodes was changed using custom JavaScript functions (indicated by temperature and humidity functions in flow diagram). These messages in desired JSON format were forwarded to an MQTT broker. Additional debugging nodes were used (green nodes) to see the payload in the gateway at various stages.

6.2.2 Payload decoding and modification

Message Payload modification in Gateway using Node-RED

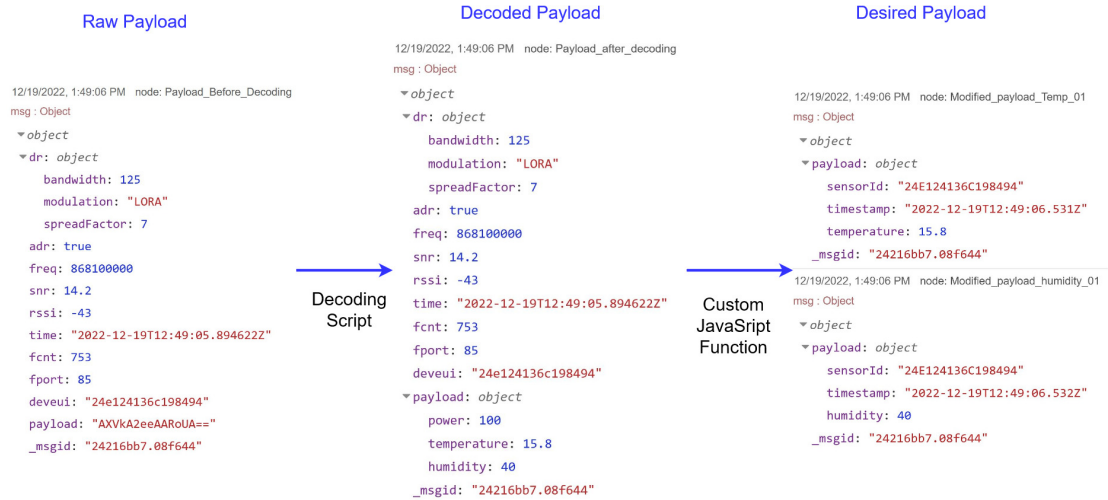
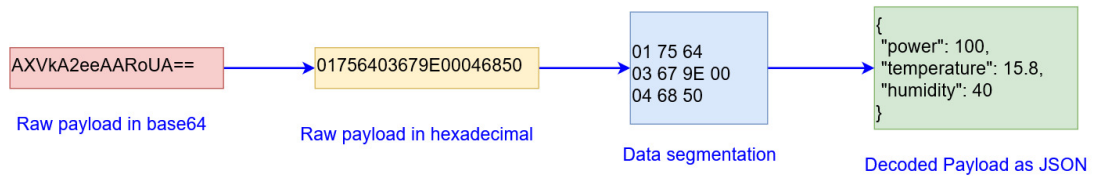


Figure 6.9: Payload decoding and modification using Node-Red

The payload was decoded and its data structure was modified into the desired format by using decoder scripts and custom JavaScript functions. Figure 6.9 depicts the data structure of the message sent by gateway. Notice that this JSON contains a field "payload" with its value in base64 format. This payload was decoded and also the data structure of the message was modified from one single JSON to two smaller JSON. These modified JSON messages (indicated by desired payload in figure) have completely different data structure from the original message.



Payload decoding logic for Milesight's EM310 Temperature humidity sensor

Channel Id	Channel type	Channel value	Value	Description	Unit
01	75	64	100	Battery level	%
03	67	9E 00	15.8	Temperature	°C
04	68	50	40	Relative Humidity	%

Figure 6.10: Payload decoding logic for temperature humidity sensor

The logic and mechanism of the decoding this raw payload is explained with the help of figure 6.10. The sensor's raw payload is encoded in base64 format. To obtain a human-readable JSON payload, the raw payload needs to be converted to hexadecimal format and segmented according to the manufacturer's logic. The first value is channel Id, second value is channel type and the third value is actual sensor data. These channel Ids and channel type represents the various variables decided by the manufacturer. Note that each manufacturer uses a different decoding logic for their sensors. For example 9E 00 converted

to decimal here means 15.8. Relative humidity value is decimal value divided by 2. Here only one such decoding mechanism is explained with an help of an example but similar logic is applied for decoding raw sensor payload from different manufacturers.

WSN Implementation using LoRaWAN, MQTT and Local Servers

7.1 Network Architecture

To address the limitations of initial tests conducted on the LoRa PHY layer, a new Wireless Sensor Network (WSN) was developed. The three major subsystems used in building this sensor network are: The sensors, the gateway and the servers. In the system architecture of this network, LoRaWAN compatible devices without any cloud services were used. Two local on premise servers were used with Mosquitto MQTT broker and other required software tools like Node-RED. Each sub-system used in the prototype WSN is discussed in following sections in detail. Figure 7.1 shows the system architecture of the WSN created for prototyping and running tests. During the prototyping the network architecture was continuously changed for tests and optimization.

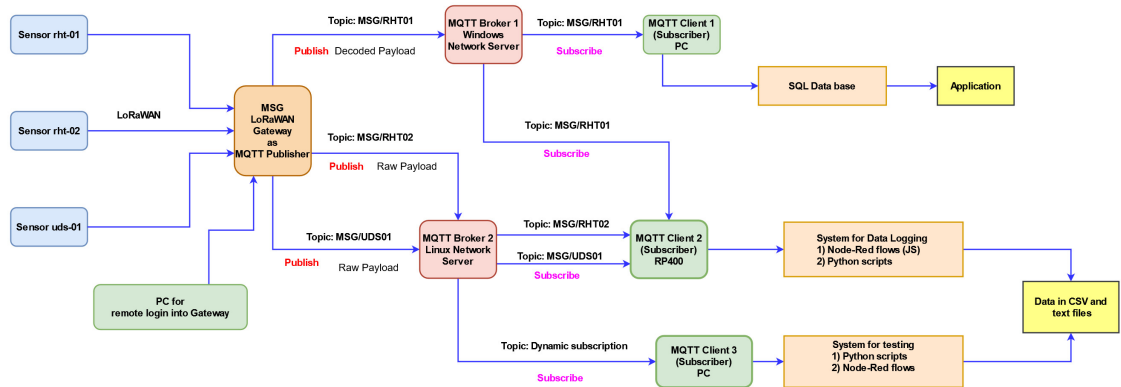


Figure 7.1: System architecture of network with local MQTT servers

7.1.1 Sensor Nodes

In this prototype network, two relative humidity and temperature sensors (indicated by rht-01 and rht-02) and one ultrasonic distance sensor (indicated by uds-01) were used. These LoRaWAN sensors were sending coded and encrypted data packets to the LoRaWAN

gateway (indicated by MSG). Sensors were configured to send data packets at different polling intervals ranging from 2-5 minutes during the tests. This polling interval was changed several times during the tests. LoRaWAN transmission frequency at which the sensors were sending data packets varied from 868.1 MHz to 868.7 MHz during the tests. This LoRa transmission frequency changes for each uplink message randomly for every device. See figure 7.2 for sensor details:

Sensor Name	Description	Payload Information	Application Information
rht-01	Temperature and relative humidity sensor Manufacturer: Milesight	LoRaWAN data packet decoded in the gateway then transmitted to MQTT server as a JSON message	Demo Appltacion 01 Topic: MSG/RHT01 MQTT Server: MQTT Server 1 (Windows)
rht-02	Temperature and relative humidity sensor Manufacturer: Milesight	Raw data packets transmitted to the MQTT server without doing any modification in gateway	Test Appltacion 02 Topic: MSG/RHT02 MQTT Server: MQTT Server 2 (Linux)
uds-01	Ultrasonic distance sensor Manufacturer: Milesight	Raw data packets transmitted to the MQTT server without doing any modification in gateway	Test Appltacion 03 Topic: MSG/UDS01 MQTT Server: MQTT Server 2 (Linux)



Temperature Humidity Sensor



Ultrasonic Distance Sensor

Figure 7.2: Temperature-humidity and ultrasonic distance sensor nodes

7.1.2 Gateway: As an MQTT Client

For this WSN a Milesight's UG65 LoRaWAN indoor gateway was used. The gateway was connected to the backhaul internet by an Ethernet cable. Login into LoRaWAN gateway was done over WLAN network, protected by username and password authentication. The login access, creating applications, device management and other important operations were performed by using the WebGUI. In this prototype network LoRaWAN gateway act as an MQTT publishing client. Several applications can be created inside the gateway (Co-located LoRaWAN application server) that acts as an individual MQTT publishing client. So this gateway worked as if multiple MQTT publishing clients were running on the same physical machine. This gives the advantage of sending data packets of different applications in gateway to different external MQTT Servers from one physical machine.

Applications in Gateway: Three different applications were created inside the LoRaWAN gateway for the sensors using WebGUI. Each application created inside the gateway acted as a distinct MQTT publishing client. These applications were pointed to separate MQTT servers with usual MQTT connection parameters like broker address, port number, and authentication details. See figure 7.3 for applications created in the gateway. Note that an application usually has multiple sensors associated with it. In this gateway, one client application could publish data to multiple MQTT topics while being connected to the same broker. However, one MQTT client application in the gateway can send data to only one MQTT broker address.

Downlink Messages: It was also possible to send downlink messages from the gateway to the target LoRaWAN devices. This ability to send downlink messages from the gateway provides a very useful way for upgrading firmware or re-configuring sensors without going to the actual site of the installed sensors. Downlink messages from the gateway to the sensors are rare and usually sent only for changing the polling frequency and for the firmware upgrade of the sensors. The downlink payload can be sent in hex or base64 format. Network traffic between the gateway and sensors is asymmetric and it is dominated by the uplink messages.

7.1.3 MQTT Servers

Two separate MQTT servers (MQTT Brokers) were created by installing Mosquitto MQTT broker on the machines. One dedicated Windows machine was used as a demonstration system and another Linux machine (A small Raspberry Pi Single Board Computer) was used as the test server. Initial tests were conducted without using any authentication between Gateway and MQTT servers. On both the machines, communication between the gateway and MQTT brokers was on port:1883 (port for unsecured connections) and both the servers were continuously running. Later, tests were conducted with authentication. For secured connections, MQTT port 8883 was used.

7.1.4 MQTT Subscribing Clients

During the prototyping, multiple MQTT Clients were used for visualizing and logging the sensor data. These clients ranged from resourced-constrained Raspberry Pi to PCs. Both Node-RED flows and Python scripts were used for the subscribing clients. Paho MQTT client library was used to write Python scripts for the subscribing clients. These Python scripts subscribed to various topics on which the applications running in the gateway were published through the broker. These scripts also logged the incoming sensor data in simple text files without doing any payload decoding and modification in the data structure of incoming messages in JSON format. Subscribing clients created by using Node-RED flows were more sophisticated compared to the Python scripts. These clients were able to do packet decoding, make changes to the data structure of the message payload, and conversion of the JSON to CSV files. When tests were running for multiple days, the sensor data was logged to the local PC and Raspberry Pi (indicated by RP400 in system architecture) using the Node-RED flows (figure 7.5)

7.1.5 Information flow in the network

Sensor data from rht-01 was published on the Windows MQTT server to the topic: MSG/RHT01. Data packets from sensors rht-02 and uds-01 were published to the topics MSG/RHT02 and MSG/UDS01 on the Linux server respectively. For these two sensors, no decoding script was used in applications running in the gateway (indicated by Test applications 01 and 02). MQTT server details like broker address, MQTT port, MQTT topic, and authentication details were configured inside the application associated with the sensors. Sensors send the coded data packets to the gateway in base64 format to keep the packet size smaller. Payload decoding and modification for these two sensors were done on the subscribing client. See Figure 7.3.

Gateway forwards the data packets received from the sensors to the MQTT servers as a

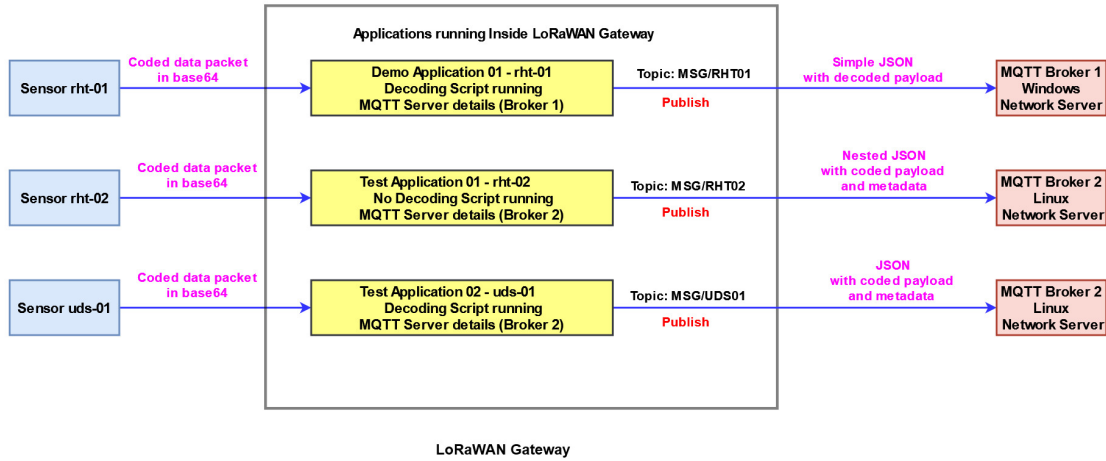


Figure 7.3: MQTT client applications inside gateway and flow of information in the network

nested JSON. When the gateway forwards a message to the MQTT server, it adds additional information (metadata) on top of the actual sensor payload (sensor readings/values in base64 format). This metadata is very useful for device management, testing, and troubleshooting. It contains information about the device (DevEUI, gateway EUI, timestamp), LoRa radio link parameters like LoRa transmission frequency, RSSI, LoRa SNR, Band Width (BW), Spreading Factor (SF), Code Rate (CR) and MQTT message topic on which the message was published. By default (without any decoding scripts running inside the gateway application) gateway sends the message (raw sensor data + metadata) as a nested JSON object. Figure 7.4 shows a raw message payload from the ultrasonic distance sensor as a nested JSON.

In the application running inside gateway, a decoding script (written in JavaScript) was used for the sensor rht-01 from the manufacturer of the sensor. The purpose of the decoding script was to decode the message payload, convert it from base64 to simple human readable JSON text. This computation was done on the edge to reduce the complexity associated in processing the nested JSON which the gateway sends to the MQTT server. Decoding script varies for each sensor depending on the payload of the sensor. The actual payload which the sensor sends also varies depending on the firmware from manufacturer. For example sensor rht-01 sends the battery level in payload every 6 hours but uds-01 sends battery level with each message. For more details on how this decoding is done and how the payload data structure is modified see figure 7.5.

7.2 Data Logging System of network

This section explains the working of the data logging system with an example from an ultrasonic distance sensor (uds-01) by using Node-Red.

Gateway forwards the data packet from the ultrasonic distance sensor (uds-01) as a nested JSON to the MQTT broker (Linux network server). Received data in the nested JSON structure is not suited for SQL / relational databases in data analytics. MQTT broker cannot change the data structure of the received messages on its own. This has to be done either with the publishing client or the subscribing client. So in one of the subscribing clients (MQTT Client 2), a Node-RED flow was created for this purpose. The Node-RED flow in 7.5 performs the following tasks:

```

1  {
2    "topic": "MSG/UDS01",
3    "payload": {
4      "applicationID": "16",
5      "applicationName": "Raspberry_Pi_400_UDS",
6      "data": "AXVjA4J+CGQAAA==",
7      "devEUI": "24e124713c018455",
8      "deviceName": "uds-01",
9      "fCnt": 2007,
10     "fPort": 85,
11     "rxInfo": [
12       {
13         "altitude": 0,
14         "latitude": 0,
15         "loRaSNR": 13.5,
16         "longitude": 0,
17         "mac": "24e124ffef57bd0",
18         "name": "Local Gateway",
19         "rssi": -55,
20         "time": "2023-02-06T12:56:18.062038Z"
21       }
22     ],
23     "time": "2023-02-06T12:56:18.062038Z",
24     "txInfo": {
25       "adr": true,
26       "codeRate": "4/5",
27       "dataRate": {
28         "bandwidth": 125,
29         "modulation": "LORA",
30         "spreadFactor": 7
31       },
32       "frequency": 868300000
33     }
34   },
35   "qos": 0,
36   "retain": false,
37   "_msgid": "a64d9e13d55be44a"
38 }

```

Figure 7.4: Raw message payload from ultrasonic distance sensor

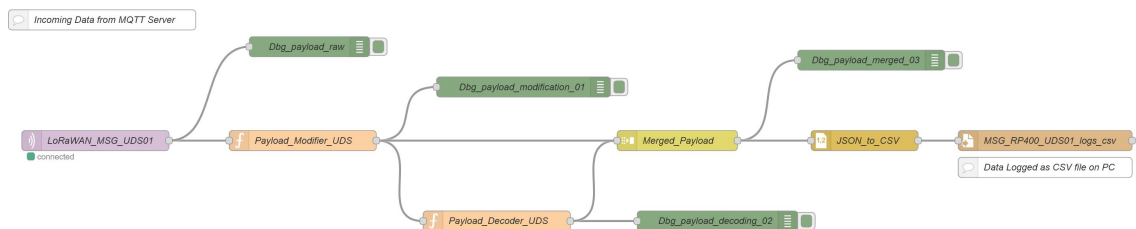


Figure 7.5: Node-RED flow for sensor payload decoding and data logging

- MQTT subscribing client is subscribed to the topic on which ultrasonic distance sensor is publishing (Topic: MSG/UDS01). MQTT subscribing client (client 2) receives the raw payload with other metadata from gateway as a single nested JSON.
- Function node "Payload Modifier UDS" is a JavaScript function that modifies the message payload and changes the data structure of the received raw message to a simple JavaScript object. It also removes the certain fields from the JSON. Output of this function node is an object that contains coded sensor data in base64 format as well as fields like timestamp, Topic, LoRa data like SNR, RSSI, DevEUI, Gateway EUI etc. This output object is fed into another function node "Payload Decoder".
- Function Node "Payload Decoder" decodes the sensor data from coded base64 format to human readable text in JSON using a decoding script written in JavaScript (from the sensor manufacturer). After decoding, the output of this node contains the actual sensor data like distance, tilt level and battery level inside an object in plain text. This output is fed to another node for merging the payloads. Note that decoding script used to decode the sensor data varies from sensor to sensor and also with manufacturer of the sensor.
- "Merged payload" node combines the output of the "Payload Modifier UDS" and "Payload Decoder" into a single object. Now this merged object has a simplified data structure which contains only the desired parameters and decoded sensor data. This data structure is suitable for storing data and further processing in relational databases.
- Output from the "Merged payload" node was converted to a CSV file and stored locally. Figure 7.6 shows the final data logged in a CSV file. This stored CSV file was used further for data visualization and analytics. Note that the figure contains the duplicate entries of the logged data. These duplicate entries were later removed while doing data analytics with the help of python scripts.

1	Topic	timestamp	devEUI	deviceName	Data_raw	battery	distance	position	gatewayEUI	rss	loRaSNR	frequency
2	MSG/UDS01	2023-02-06T08:50:18.111428Z	24e124713c018455	uds-01	AXVjA4JeBwQAAA==	99	1886	normal	24e124ffef57bd0	-56	10	868500000
3	MSG/UDS01	2023-02-06T08:52:18.109342Z	24e124713c018455	uds-01	AXVjA4JeBwQAAA==	99	1886	normal	24e124ffef57bd0	-58	13.5	868100000
4	MSG/UDS01	2023-02-06T08:52:18.109342Z	24e124713c018455	uds-01	AXVjA4JeBwQAAA==	99	1886	normal	24e124ffef57bd0	-58	13.5	868100000
5	MSG/UDS01	2023-02-06T08:54:18.116412Z	24e124713c018455	uds-01	AXVjA4LABgQAAA==	99	1886	normal	24e124ffef57bd0	-59	13.5	868100000
6	MSG/UDS01	2023-02-06T08:54:18.116412Z	24e124713c018455	uds-01	AXVjA4LABgQAAA==	99	1728	normal	24e124ffef57bd0	-59	13.5	868100000
7	MSG/UDS01	2023-02-06T08:56:18.11149Z	24e124713c018455	uds-01	AXVjA4LCBgQAAA==	99	1728	normal	24e124ffef57bd0	-59	12	868500000
8	MSG/UDS01	2023-02-06T08:56:18.11149Z	24e124713c018455	uds-01	AXVjA4LCBgQAAA==	99	1730	normal	24e124ffef57bd0	-59	12	868500000
9	MSG/UDS01	2023-02-06T08:58:18.115359Z	24e124713c018455	uds-01	AXVjA4LCBgQAAA==	99	1730	normal	24e124ffef57bd0	-62	14	868100000
10	MSG/UDS01	2023-02-06T08:58:18.115359Z	24e124713c018455	uds-01	AXVjA4LCBgQAAA==	99	1730	normal	24e124ffef57bd0	-62	14	868100000

Figure 7.6: Data logged from ultrasonic distance sensor after modification

7.3 Network security implementation

LoRaWAN sensors and Gateway have 2 layers security in between them. The communication between them uses 128 bit AES encryption. Sensor and associated application inside gateway are fed with the same 128 bit application key. This AppKey generates two keys - Network Session Key and application session key. These keys encrypt the communication between LoRaWAN Gateway and the sensors. The login into the gateway is password protected. The connection between gateway and the MQTT server is also password protected and secured by TLS encryption. Same security exists between various MQTT subscribing

clients and the MQTT broker. In this prototype network there were several layers of security depending on the interaction on sub-systems with each other. These security layers are shown in figure 7.7

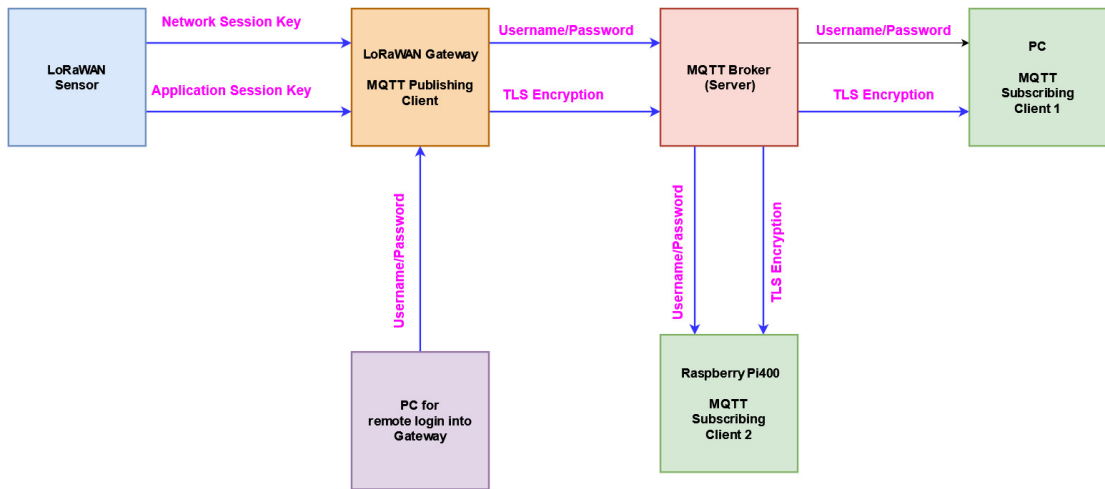


Figure 7.7: Network Security with local MQTT server and clients

Results and Analysis

This chapter covers the Exploratory Data Analysis (EDA) of sensor data collected from the tests. This EDA is done by using popular Python libraries for data processing and visualization like Pandas, Numpy, Seaborn, and Matplotlib.

8.1 EDA of temperature humidity sensors

For sensor rht-01, decoding of the data packets from base64 format to human-readable plain text was done in the gateway itself. Milesight gateway provides an option to run the decoding script in the application associated with the sensor. The packet forwarded by the gateway was a simple JSON message containing temperature and humidity values in key-value pairs. The tests ran between 01.02.2023 - 06.02.2023 and the sensor was polled every 3 minutes. A total of 2264 samples were collected and stored in CSV files. Note that no metadata related to LoRa parameters was sent by the gateway to the MQTT broker.

Data collected from the Milesight temperature humidity sensor (sensor named rht-01) is shown in figure 8.1. This sensor was placed near the measurement machines. The cyclical sharp rise in temperature and fall in humidity is due to turning the heat pump "on" during day time. This negative correlation between temperature and humidity is depicted by figure 8.2. It can be noticed clearly that as the temperature rises humidity falls and vice versa.

For sensor rht-02 the 689 samples were recorded between 31.01.2023 - 01.02.2023. The sensor was polled every 2 minutes. For this sensor, the first few entries of recorded data are shown in 8.3. Note that no decoding script is used for this sensor and raw data packets as well as metadata from the gateway were stored in CSV files.

LoRa frequency channel utilization is shown in figure 8.4. All the channels (868.1, 868.3 and 868.5 MHz) show nearly equal usage with 868.1 MHz having highest utilization of around 35 %. LoRa SNR value is shown in figure 8.5 as a violin plot. It varied from a minimum value of 8.2 to maximum of 14.5. The mean LoRa SNR was 12.37. LoRa Received Signal Strength Indicator (RSSI) value varied between -40 dBm and -91dBm

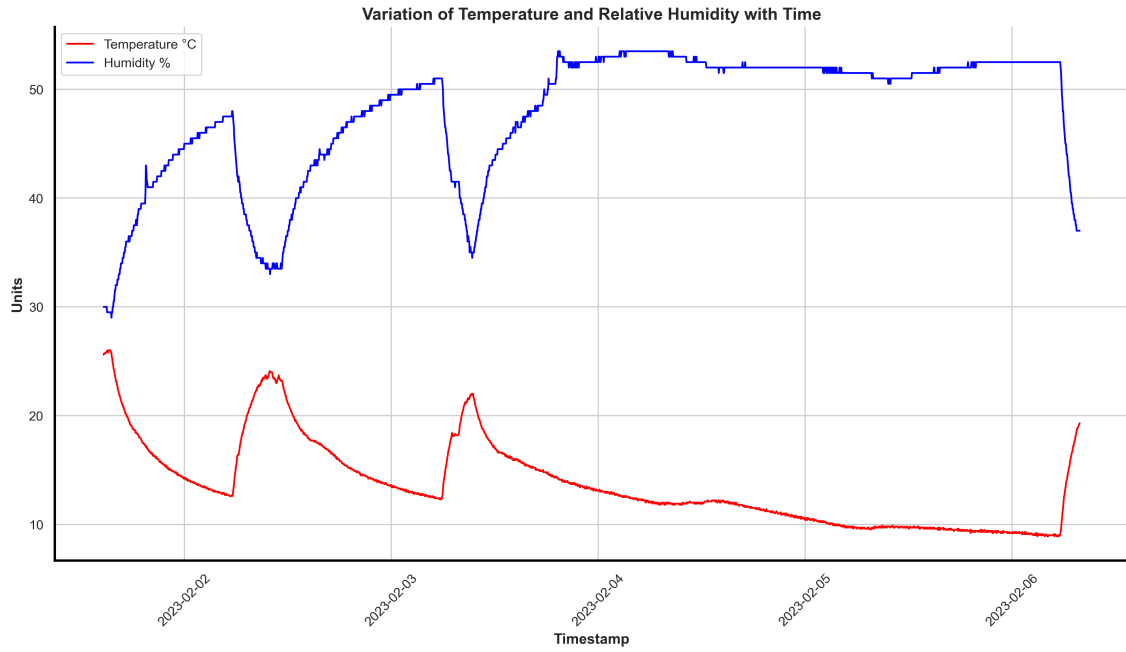


Figure 8.1: Data collected from temperature humidity sensor rht-01

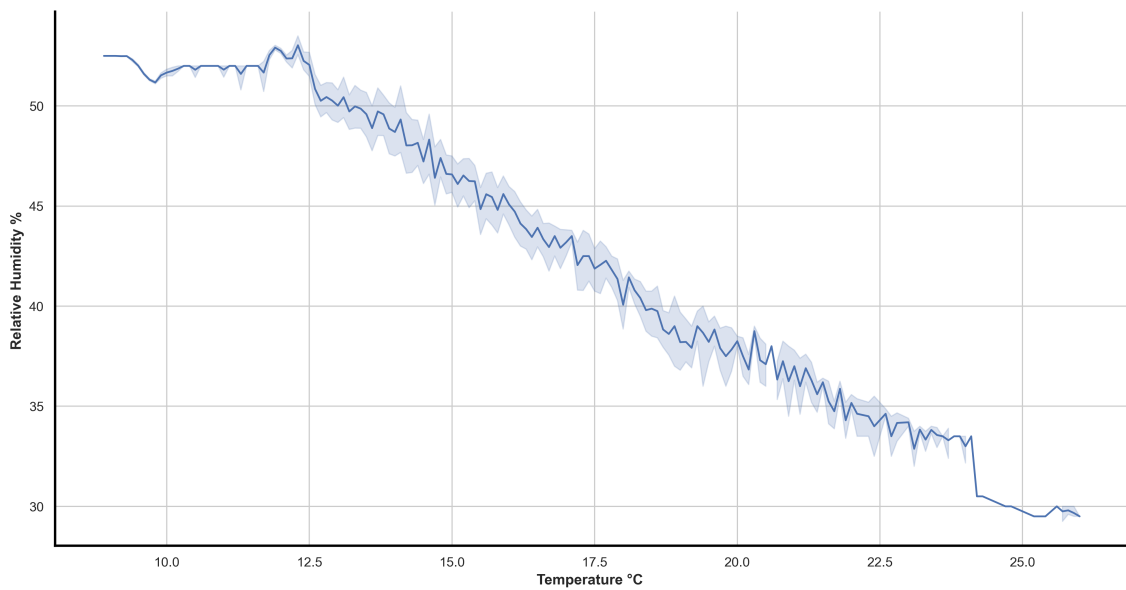


Figure 8.2: Temperature humidity correlation

	Topic	timestamp	Data	devEUI	deviceName	gatewayEUI	rssi	IoRaSNR	frequency
0	MSG/RHT02	2023-01-31T15:27:47.584032Z	A2fEAARoVg==	24e124136c220737	rht-02	24e124ffef57bd0	-52	13.2	868300000
1	MSG/RHT02	2023-01-31T15:29:47.586139Z	A2fEAARoVg==	24e124136c220737	rht-02	24e124ffef57bd0	-53	14.0	868100000
2	MSG/RHT02	2023-01-31T15:31:47.586622Z	A2fEAARoVQ==	24e124136c220737	rht-02	24e124ffef57bd0	-49	11.5	868500000
3	MSG/RHT02	2023-01-31T15:33:47.593672Z	A2fEAARoVQ==	24e124136c220737	rht-02	24e124ffef57bd0	-52	13.8	868300000
4	MSG/RHT02	2023-01-31T15:35:47.582386Z	A2fEAARoVQ==	24e124136c220737	rht-02	24e124ffef57bd0	-46	13.5	868300000

Figure 8.3: Data collected from temperature humidity sensor rht-02

with the mean RSSI value of -45.6 dBm as shown in figure 8.6 as a violin plot. Note that these unusually high values of SNR and RSSI is due to the fact that sensors were not very from the gateway (about 100m distance) during these tests.

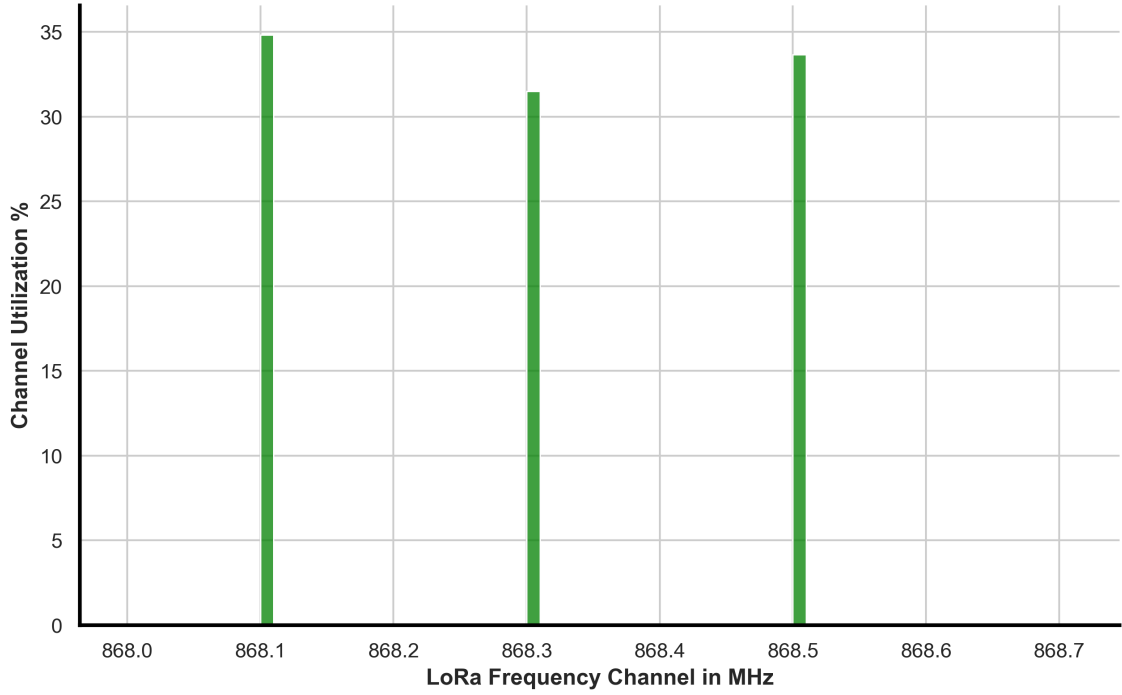


Figure 8.4: LoRa frequency channel utilization for sensor rht-02

8.2 EDA of data collected from ultrasonic distance sensors

For the ultrasonic distance sensor, 163 unique data samples were recorded on 06.02.2023 with a polling frequency of every 2 minutes. The data recorded in CSV can be seen in figure 7.6. This sensor gives the distance from the obstacle in mm and also the position of the sensor if it is normal or tilted. For uds-01, payload decoding was done on the subscribing MQTT client. Figure 8.7 shows the distance and position data recorded from the ultrasonic distance sensor.

For sensor uds-01, frequency channel of 868.5 MHz was utilized most at 38 % as shown in figure 8.8. LoRa SNR varied between 8.2 and 14.2 with a median value of 13.5 as shown in figure 8.9. RSSI value varied between -84 dBm and -49dBm having a median value of -57dBm as shown in figure 8.10.

These tests explored the possibility of doing some computation (sensor payload decoding in gateway) on the edge. From these tests it can be concluded that sensor data can be sent successfully using commercial LoRaWAN compatible sensor nodes using MQTT and local on-premise servers. However there were certain limitations and challenges associated with this network architecture for scaling. These are discussed in the following sections.

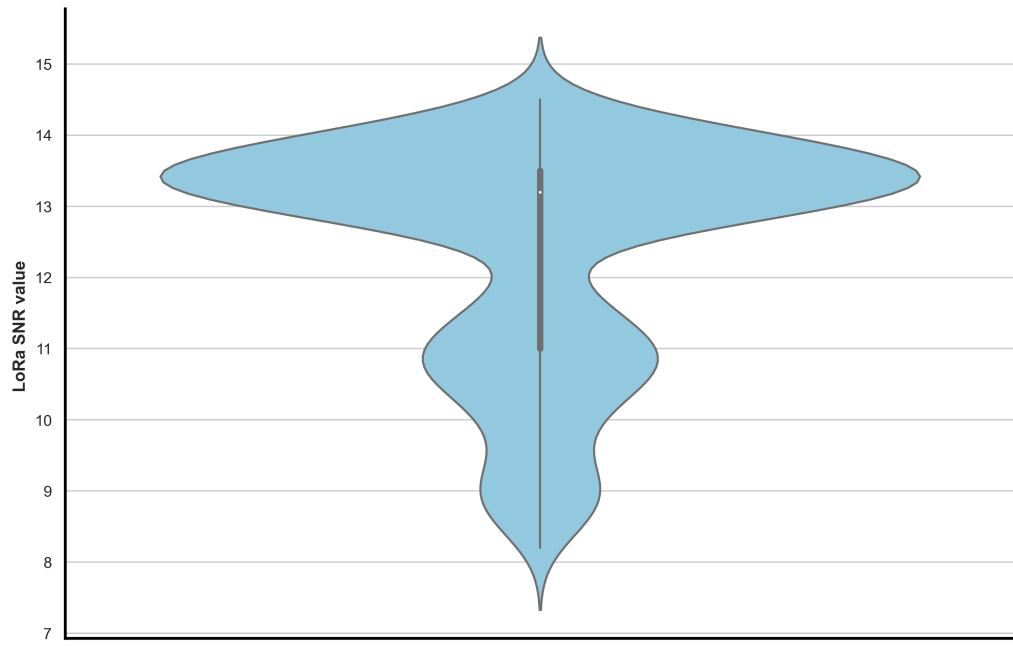


Figure 8.5: LoRa SNR variation for sensor rht-02

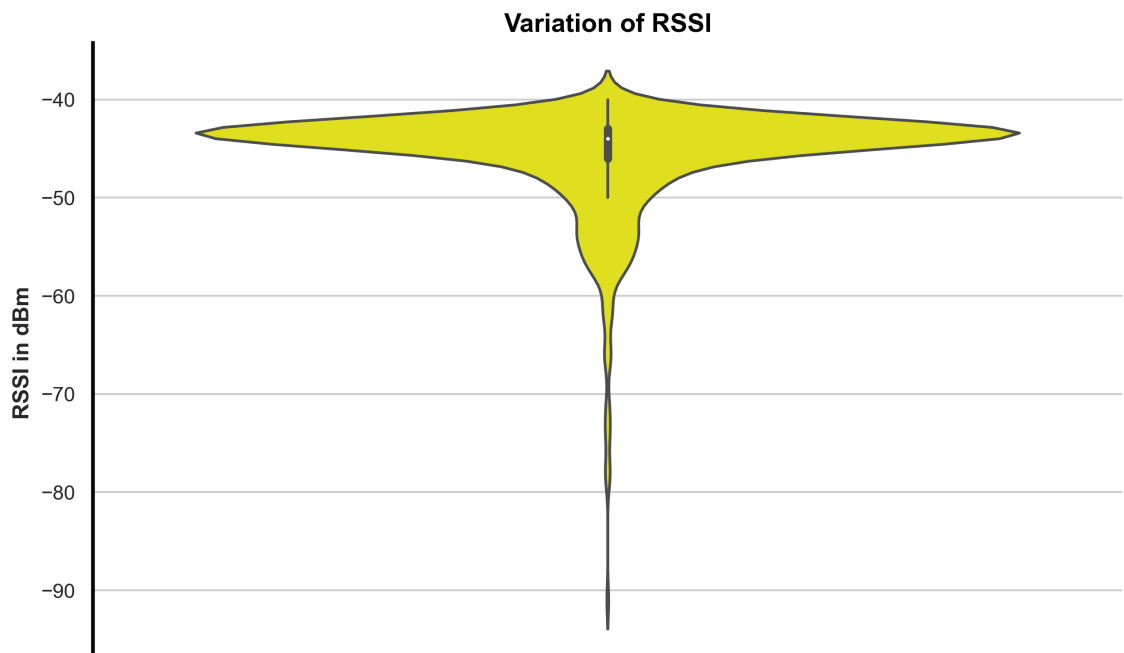


Figure 8.6: RSSI variation for sensor rht-02

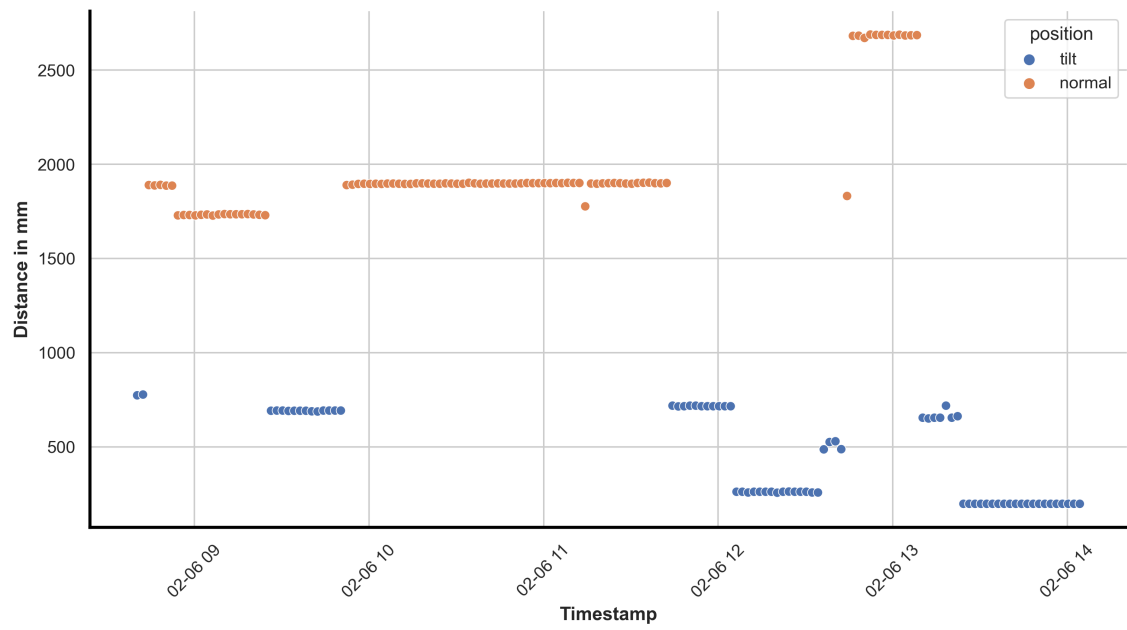


Figure 8.7: Data collected from ultrasonic distance sensor uds-01

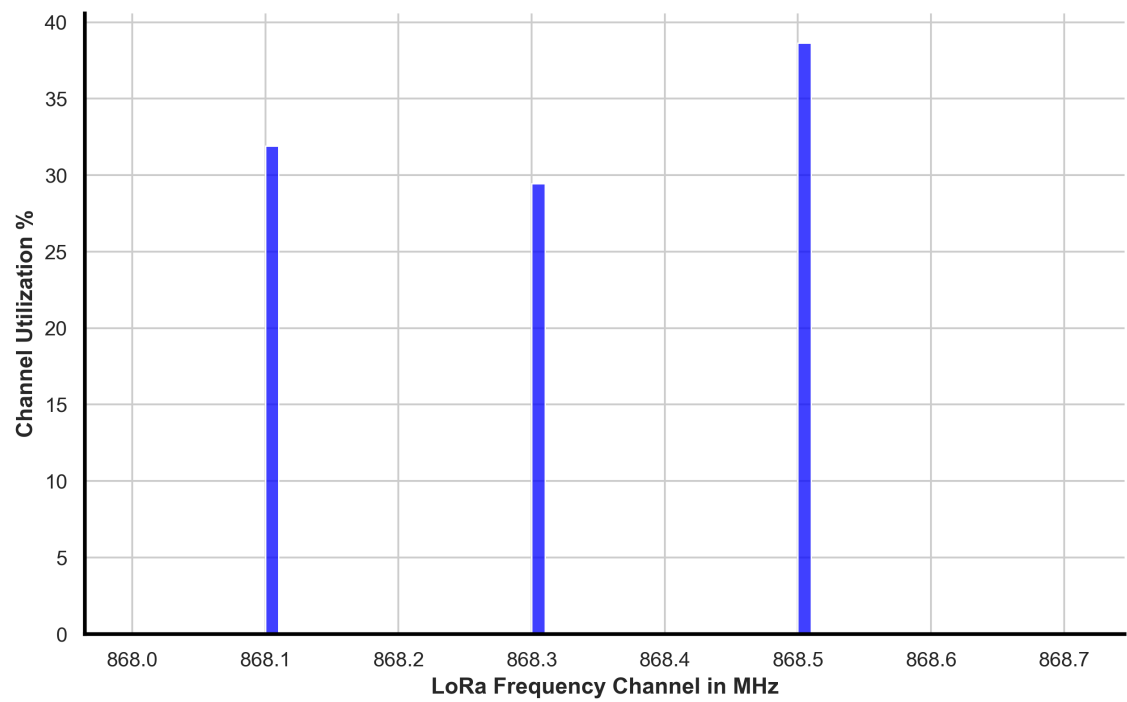


Figure 8.8: LoRa frequency channel utilization for sensor uds-01

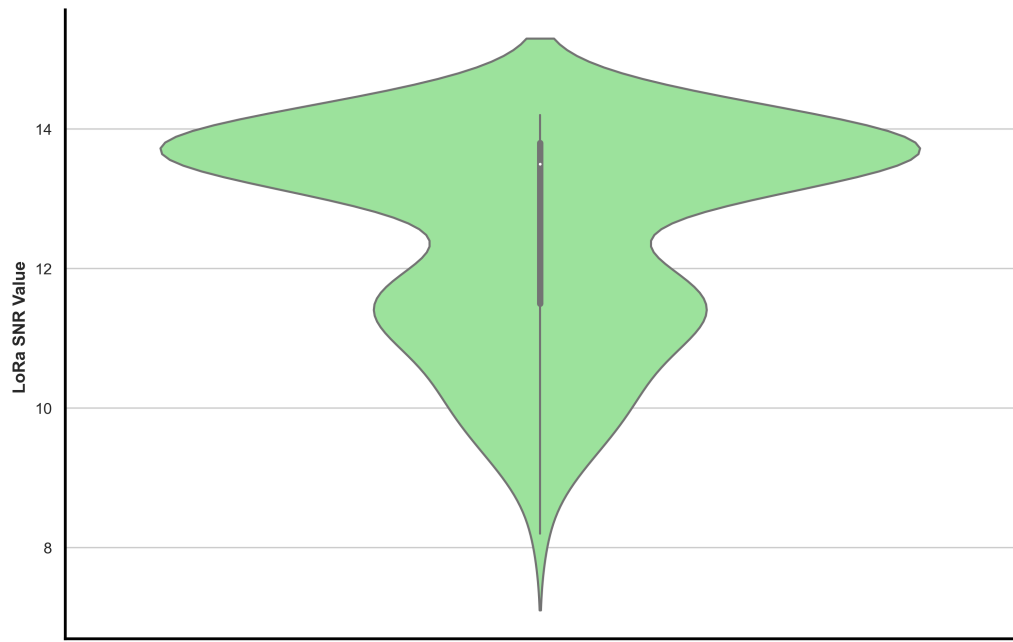


Figure 8.9: LoRa SNR variation for sensor uds-01

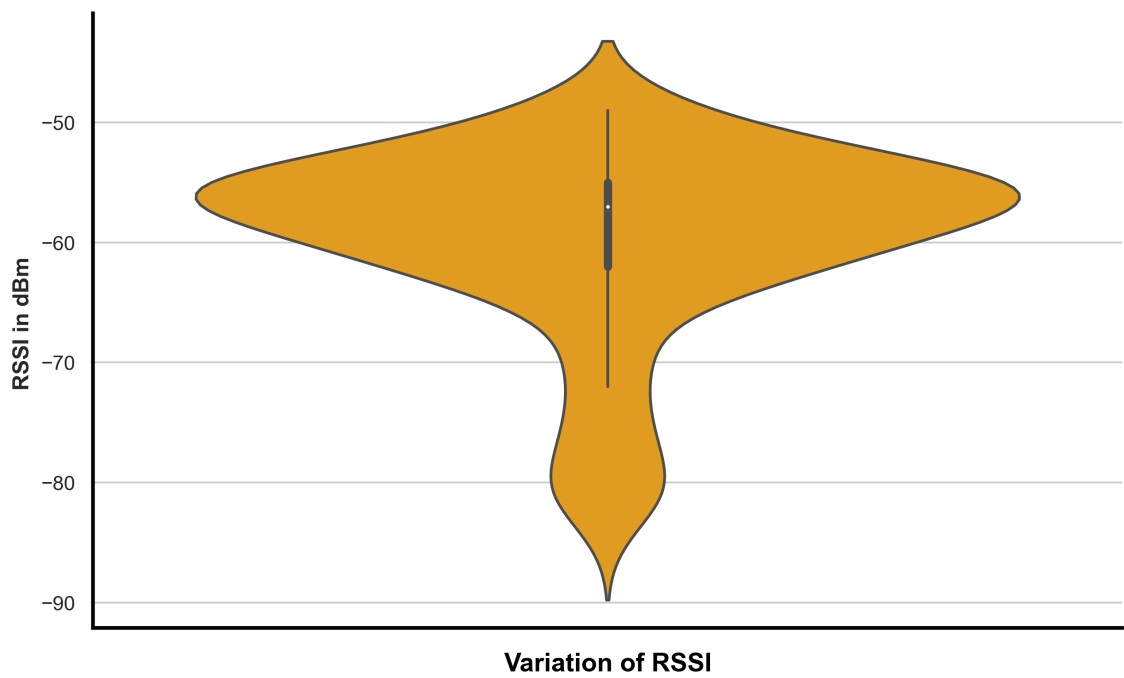


Figure 8.10: RSSI variation for sensor uds-01

8.3 Analysis

This section summarizes the key findings and lessons learned from prototyping the WSN. It includes the key challenges and limitations of the various subsystems in the sensor network.

8.3.1 Challenges with message payload decoding

One of the key tasks in successfully running a LoRaWAN-MQTT based wireless sensor network is to decode the message payload and modify the data structure of the incoming messages from the sensors. These modified messages should be in the desired JSON suitable for storage in relational database in servers and further data analytics. This modification and payload decoding can be done at several points in the network. Message payload can be modified either in the gateway itself via Node-RED flows/Software Development Kits (SDKs) or in the local/cloud server. Both of these design choices have certain advantages and limitations. For example it is possible to write a custom JavaScript functions in gateway application or via Node-red flows in Milesight's UG65 gateway. This JavaScript function can decode the the message payload and modifies the message data structure into a simple JSON in the gateway itself instead of doing it on the subscribing client! (In Milesight UG65 gateway used for prototyping). Note that all gateways do not provide the ability to run decoding scripts and Node-Red flows.

There are several limitations associated with making these modifications in applications running in the gateway. A custom JavaScript function written in application inside gateway for payload decoding works only for the sensors belonging to same type and from the same manufacturer (say all ultrasonic distance sensors in one gateway application). This decoding becomes complicated when the same type of sensors are used from different manufacturer. Every manufacturer uses a different payload decoder script for their sensors. For example, one decoding script running in the gateway application for ultrasonic distance sensors will not be able to decode the message payload from the another manufacturer's ultrasonic distance sensors correctly. Additionally when the decoding script is running inside gateway, then the gateway application only forwards the sensors readings only. In this case, important metadata like DevEUI is not forwarded to the servers. This make the identification of data from which sensor it is coming from very difficult.

8.3.2 Challenges with scaling

One big advantage of doing sensor payload decoding in the gateway itself (edge computing) is that there is no additional cost associated with running these payload decoding scripts in the gateway. But when the network has a large number of sensor nodes and gateways then, sensor payload decoding and modification in gateways becomes very complicated with Node-RED message flows and sensor decoding scripts. These modifications need to be made in every gateway in the network. Moreover most gateways do not have the functionality to run these decoding scripts.

Another way of modifying data structure of message and decoding the message payload is to do it on the local server or cloud server (Microsoft Azure used in the later tests) instead of doing the same in application running in the gateway. This approach has several advantages and scales better for a large number of devices. Rather than writing custom JavaScript functions in each gateway application for decoding the message payload from various different type of devices in network, several custom JavaScript functions can be written for a various class of sensors on the subscribing client on the local/cloud servers.

Later these functions can be applied to the incoming data packets from each sensor node accordingly to achieve the desired output suitable for the back-end database. This also creates a single point source for making changes to the network configuration (with regards to payload decoding and modification) instead of doing it in each gateway separately. To address these challenges, improved network architecture having a combination of local servers and cloud services were used in the later tests as discussed in next chapter.

Prototype Network with Cloud Server

The computing infrastructure required to run the IoT platform can be hosted by on-premise local servers, by a commercial cloud service provider or by a combination of both. The choice between different solutions depends on the application specific needs. The most important factors in making this design choice are listed below:

- **Ease of infrastructure setup and maintenance:** It is relatively easy to set up the back-end infrastructure with cloud service provider like AWS and Microsoft Azure as compared to setting up the locally hosted infrastructure. On-premise solution require skill to setup and maintain.
- **Scalability:** Cloud systems are highly scalable as per the computing demand. Infrastructure can be down-scaled in time of low demand. Scalability of On-premise hardware is limited.
- **Cost:** The upfront costs associated with on-premise solution is quite high but the operational expenditure is low. In cloud services, the cost model is pay as you use which is very cost effective.
- **Cybersecurity and risk compliance:** Cloud service providers offers a higher level of security. On-premise solutions are more vulnerable to cyber threats and requires specialized IT skill to manage and mitigate the security incidents. Note that on-premise solution offers more control over data privacy for highly regulated application involving Personally Identifiable Information (PII).
- **Integration with other applications:** Sometimes the biggest deciding factor to choose between cloud and on-premise solution is the feasibility of smooth integration with existing infrastructure. The cost savings are huge in development time and system integration with cloud.

9.1 Microsoft Azure cloud services

Microsoft Azure cloud was used for the back-end infrastructure of the IoT platform during the research. Fully managed cloud services were used instead of hosting the virtual machine. Azure provides cloud computing platform for applications like storage, computation and IoT device management. It offers different subsystems that can be used to build an end to end integrated IoT system. Some of these are mentioned below:

- Devices connectivity management for smart sensors, industrial IoT Gateways, smart PLCs etc by Azure IoT hub.
- Cloud service which performs the functions like: receiving data, device management and authentication, downlink communication for software upgrades.
- Cloud hosted SQL databases for sensor data storage.
- Stream processors which consumes the incoming sensor data, process and stores it in databases.
- Data visualization and user interface by application like Azure time series insights and Power BI

Note that all of these subsystems need not be used for creating the IoT platform. Some of the tasks can be performed on small resource-constrained servers. Figure 9.1 shows an overview of the cloud services provided by Microsoft Azure that can be used for creating IoT platform.

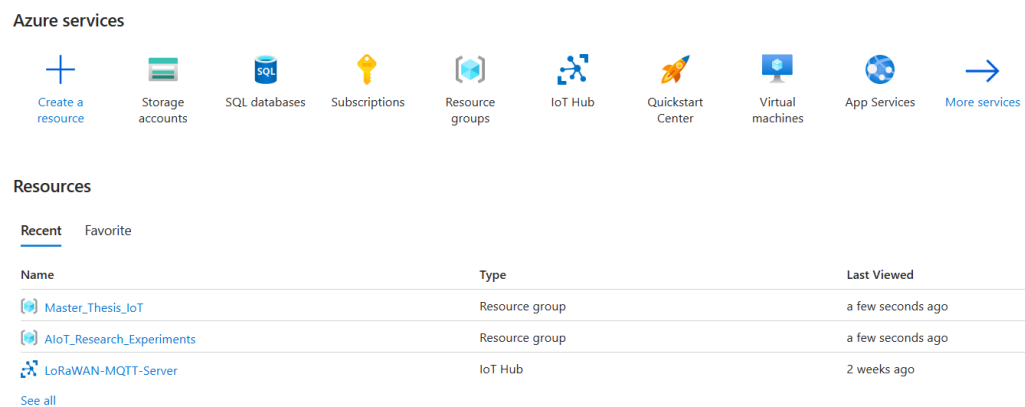


Figure 9.1: Network architecture with Azure back-end

Resource Groups: Azure provides the option to use resource groups as logical containers holding all the resources related to specific project. It can contain resources like virtual machine, SQL databases, IoT hub and APIs. It is easier to manage and organize computing resources with resource groups.

Azure IoT Hub: Azure IoT hub is a cloud service platform used for connection and management of IoT devices. It could be considered as an MQTT broker that offers both device to cloud and cloud to device messaging options. Note that Azure IoT hub does not implement the full MQTT standard like Mosquitto; see Microsoft Azure's official documentation for more details on Azure IoT hub [19].

9.2 Network I

To improve on the limitations of the previous network, PostgreSQL hosted in Microsoft Azure cloud was used for the database. In the test network, a different LoRaWAN gateway (RAK indoor gateway 7268C) was used. This gateway did not have the functionality to run decoding scripts in the gateway. So decoding scripts for sensor payload ran on MQTT subscribing client. This gateway supports global integration with only one MQTT broker. It was not possible to send the sensor data from each application to multiple MQTT servers. For the tests, a locally hosted Mosquitto MQTT broker was used. The network architecture is shown in figure 9.3. Six different sensor node were used in this network. Four different applications were created for these sensors in the gateway as shown below in the table 9.1:

Table 9.1: Sensor applications in gateway

Application Id	Name	Description	Number
01	RAK-MS-RHT	Temperature humidity sensor	2
02	RAK-MS-UDS	Ultrasonic distance sensor	1
03	RAK-IMB-PCS	People counter sensor	1
04	RAK-ES-EMS	Environment monitoring door sensor	2

Sensors rht-01, rht-02, and uds-01 are temperature humidity, and ultrasonic distance sensors from the last experiments. Sensor pcs-01 is an infrared-based people counter sensor from the manufacturer IM Buildings. It is suitable for counting the number of people going in and out of the building. Sensors ems-01 and ems-02 are door sensors that trigger a message on the opening and closing of the magnetic contact switch when the door is being opened or closed. This type of sensor is suitable for monitoring the environment in server rooms and warehouses.

Information flow: Sensor data from all the applications were sent to a single MQTT server. Clients could subscribe to the specific sensor from the chosen application using the topic structure shown in figure 9.3. In this network architecture, the MQTT topic structure was greatly simplified. MQTT clients created using Python and Node-Red were used for testing, debugging, and data logging. The main decoding scripts for sensor payload were running on PC. Decoding scripts were applied to incoming messages from the gateway by using the application ID, application name, and device ID. This simplified the process of running decoding scripts greatly. From this PC sensor data was forwarded to a PostgreSQL database hosted in Azure via HTTP requests for further consumption by API. This network was tested and it was found that all the sensors were sending data reliably to the back-end database hosted in the cloud. One of the data sets recorded during tests is shown in figure 9.2. Note the entries associated with application Id 1. This data shows that LoRaSNR is 0 and an RSSI value of -342. This is due to the fact these data packets were sent using FSK modulation at 50 KHz frequency and not using LoRa modulation. These false values are due to software incompatibility between the sensor node and gateway from different manufacturers.

1	Topic	Time_stamp	Application_Name	Application_ID	Gateway_EUI	Device_EUI	Device_Name	Data	RSSI	LoRa_SNR
2	application/4/device/a81758ffe07654a/rx	1678712605	RAK_ES_EMS	4	ac1f09fffe070717	a81758ffe07654a	ems-02	AQDyAiIDAP9ABw4vCwAAAAAsNAA8AEgA=	-29	13.8
3	application/1/device/24e124136c198494/rx	1678712707	RAK_MS_RHT	1	ac1f09fffe070717	24e124136c198494	rht-01	A2fqAARoTg==	-342	0
4	application/2/device/24e124713c018455/rx	1678712780	RAK_MS_UDS	2	ac1f09fffe070717	24e124713c018455	uds-01	AXVjA4JfBwQAAA==	-342	0
5	application/4/device/a81758ffe076551/rx	1678712791	RAK_ES_EMS	4	ac1f09fffe070717	a81758ffe076551	ems-01	AQD0AiID/wA+Bw4ICwAAACMNAQ8AEgA=	-48	11.3
6	application/3/device/0004a30b00f7eac6/rx	1678712905	RAK_IMB_PCS	3	ac1f09fffe070717	0004a30b00f7eac6	pcs-01	AgYABKMLAPfqxgABJwAAAAcAAAsACvU=	-49	11
7	application/4/device/a81758ffe07654a/rx	1678712905	RAK_ES_EMS	4	ac1f09fffe070717	a81758ffe07654a	ems-02	AQDzAiIDAP9ABw4vCwAAAAAsNAA8AEgA=	-33	10
8	application/1/device/24e124136c198494/rx	1678713007	RAK_MS_RHT	1	ac1f09fffe070717	24e124136c198494	rht-01	A2fsAARoTQ==	-342	0
9	application/2/device/24e124713c018455/rx	1678713080	RAK_MS_UDS	2	ac1f09fffe070717	24e124713c018455	uds-01	AXVjA4JwBwQAAA==	-342	0
10	application/4/device/a81758ffe076551/rx	1678713091	RAK_ES_EMS	4	ac1f09fffe070717	a81758ffe076551	ems-01	AQD1AiIDAAA/Bw4ICwAAACMNAQ8AEgA=	-47	13.5
11	application/4/device/a81758ffe07654a/rx	1678713205	RAK_ES_EMS	4	ac1f09fffe070717	a81758ffe07654a	ems-02	AQD0AiIDAQA/Bw4vCwAAAAAsNAA8AEgA=	-35	13.8

Figure 9.2: Recorded data from Network I

9.3 Network II

Sensor Network II as shown in figure 9.4 was deployed for real world tests in DHBW Heidenheim campus as a proof of concept. This network has some changes in the network architecture as compared to the original tested network. In the Network II, a small raspberry pi400 with all the necessary tools like decoding scripts, MQTT broker, subscribing client was connected to the network of the DHBW. This raspberry pi400 was sending the sensor data to the PostgreSQL database hosted in Azure. No other subscribing client was used for debugging and data logging. Also note that the gateway was connected to the DHBW network by Ethernet. Figure 9.5 shows the test version of IoT software platform created to display the sensor data on dashboard. Only temperature-humidity and people counter sensor values were displayed on the dashboard. Rest of the sensors were sending data to the cloud hosted database but it was not displayed on the dashboard.

Tests with cloud-hosted MQTT brokers: The MQTT broker can run on a cloud platform rather than being installed on a raspberry pi400 located on-site. Azure IoT hub and HiveMQ provide this functionality. A cloud-hosted MQTT broker (Azure IoT hub) was tried but integration with Azure IoT hub was not successful. So raspberry pi400 was used for running the broker for the deployed network. MQTT clients and decoding scripts were written in JavaScript instead of Python. The complete code and application were running in the containerized application using Docker containers.

When dealing with a large number of devices and messages, the operating expenses of using a cloud-based MQTT broker is a limitation. A cloud service provider like Amazon Web Services (AWS) IoT Core and Microsoft Azure IoT hub charge according to computation costs for each message sent from the gateway to the cloud. For example, the Microsoft Azure IoT hub used for tests during this project charges on the usage of D2C and C2D messages. This cost associated with operating cloud infrastructure could become a significant portion of network operating expenses. On the other hand, executing decoding scripts on the edge (small raspberry pi400) doesn't cost money since it is already owned infrastructure. Cloud-hosted brokers have the advantage of high reliability as they operate distributed broker clusters (multiple brokers). This gives system redundancy in case one of the brokers fails other brokers can take over without affecting the network. For a detailed comparison of commercial MQTT brokers see [11] [4].

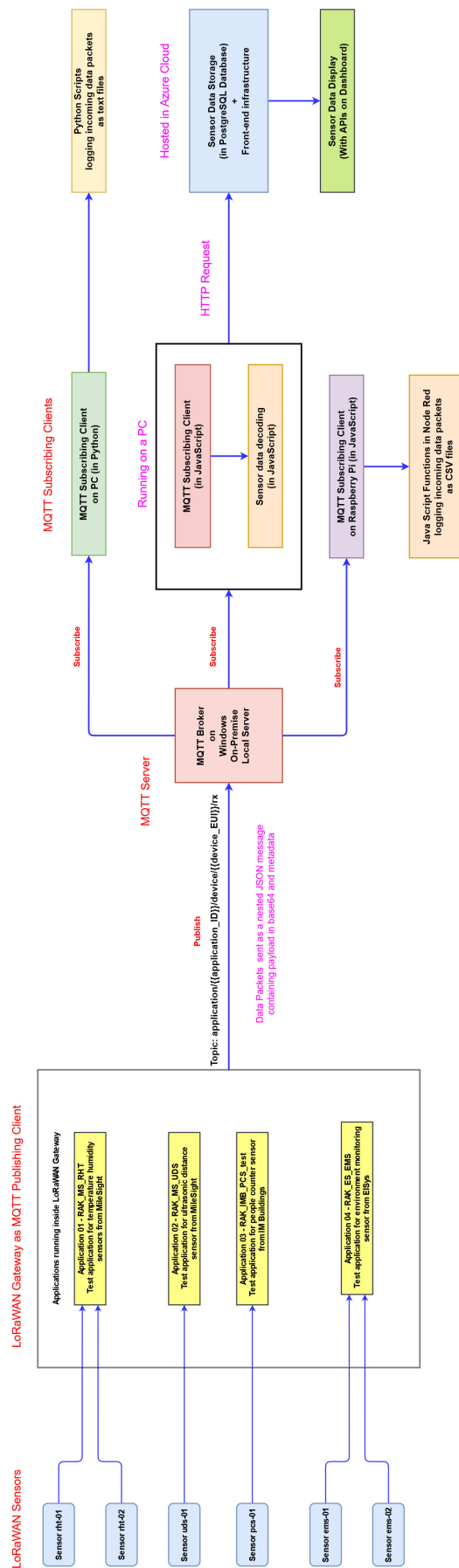


Figure 9.3: Network I architecture

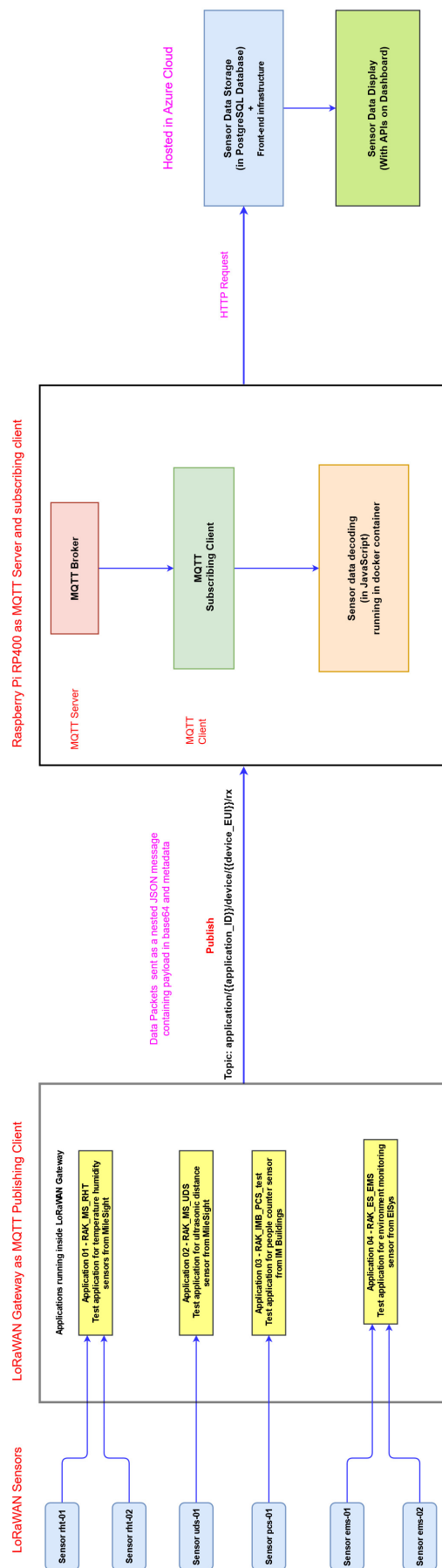


Figure 9.4: Network II architecture

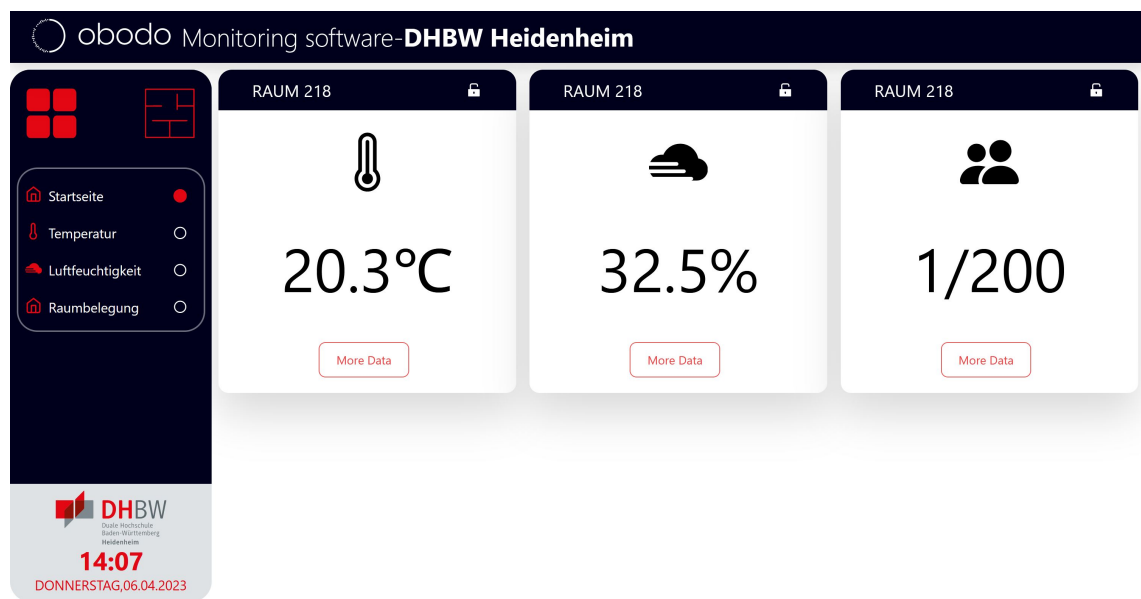


Figure 9.5: Obodo Monitoring Dashboard
Source: APS Tech Group, Obodo Platform

Summary and Conclusion

The primary focus of the research work was on creating and refining the architecture of the Wireless Sensor Networks (WSNs), establishing a system for sensor data logging, and improving the mechanisms for decoding sensor data efficiently. The research work involved conducting preliminary tests using the LoRa PHY layer, constructing various network configurations with the LoRaWAN MAC protocol, and integrating the WSNs with MQTT and cloud servers. On-premise local servers were also configured and used for the prototyping. Some of the tasks like running decoding scripts were done on the edge (resource-constrained raspberry pi400) also. The WSN was tested in real-world conditions to validate its functionality, reliability, and performance. The results of the tests showed that the WSN was able to transmit data reliably with very low power consumption. The prototype WSN is scalable, cost-effective, and suitable for applications in smart cities, industrial automation, smart agriculture, and smart buildings. The following sections discuss the current challenges in integrating the various subsystems of the WSNs based on LoRaWAN, MQTT, and cloud computing. The possible areas of improvement and suggestions for improving the system integration for future WSNs based on LoRaWAN, MQTT, and cloud computing is also discussed in the following sections.

10.1 Challenges and scope of future work

This section summarizes the findings from the tests and experiments about the LoRaWAN network equipment and the challenges in its integration with cloud infrastructure. These challenges create the scope of future research work in creating scalable LoRaWAN-MQTT-Cloud-based WSNs.

10.1.1 Challenges with the LoRaWAN gateways

- **Gateway Output JSON:** Currently for the commercial LoRaWAN gateways sold in the market, there is no use of a single standard data structure for the output message of the gateways. Gateways from different manufacturers provide different level of information for metadata in a non-standard JSON message. In the output JSON

messages from different gateways, the level of nesting and information hierarchy is different. This makes it very difficult to create a back-end database that can work with all gateways and accommodate the incoming telemetry data in the existing database. In other words it is very difficult to create a plug and play solution in which any gateway can be integrated with the existing back-end infrastructure of the IoT platform.

- **MQTT Integration in gateway application:** In the applications running inside the gateway, there is a lack of standardization on how many brokers (MQTT Servers) the gateway can forward the sensor data. For some gateways, there is scope for global integration with only one MQTT server (for example RAK gateway 7268C). For other gateways, it is possible to send data of each LoRaWAN application to several MQTT Servers. This global integration with only one MQTT broker creates a single point of failure for the whole cluster of sensor nodes if the gateway integration fails with the desired MQTT server. Also, there is a lack of fail-safe redundancy mechanisms to forward the messages to other MQTT servers in case of a connection failure with the existing MQTT server.
- **MQTT client information control level:** Gateway acts as an MQTT client. All gateways do not provide the same level of control over the MQTT client functionality. Gateway Web GUI makes the configuration of MQTT clients much easier as compared to writing MQTT clients from scratch using Python or JavaScript. This ease of configuration comes at the cost of losing control over what a user can do with the gateway as an MQTT client. The user loses some control over trivial things like the ability to choose the MQTT version, available options for SSL/TLS, the ability to upload the signed security certificates, and the level of MQTT topic (Join Topic, Up-link Topic, downlink topic, acknowledgment topic)
- **Remote access to gateway:** Most commercial gateways are accessed with a default IP address on a local network (192.168.xxx.xxx) using secure login with a PC/laptop. This default IP address to access the gateway via web GUI is configured by the gateway manufacturers and cannot be changed easily. It is very difficult to access the gateways from outside the network behind the firewall for making configuration changes to the LoRaWAN network remotely. Mechanism with the current browser-based Web GUI to do this task is quite challenging and sometimes requires the use of an external Virtual Private Network (VPN) or the device management/cloud service from the gateway manufacturers. This creates the challenge of being physically present on the site where the gateway is installed or remote access to the PC connected to the same network if some LoRaWAN network changes need to be made. Also sometimes users are not willing to give remote access to the PC on their networks due to security and privacy concerns.

10.1.2 Challenges with LoRaWAN sensor nodes

- **Sensor payload decoding scripts:** The data packet size (Payload size) is limited in LoRaWAN so sensor manufacturers use clever techniques to compress the information transmitted in sensor payload. To decode this payload and extract any meaningful information requires the use of decoding scripts. Most of the manufacturers provide decoding scripts written in JavaScript only. These decoding scripts can run on some gateways' JavaScript engine but for some gateways there is no option to run these decoding scripts. So while running the decoding scripts on back-end servers, a JavaScript framework is required. It is also difficult to use the decoding scripts written in JavaScript with MQTT Clients written in Python, and back-end framework also in python (For example Django).

- **Use of battery in sensor nodes:** LoRaWAN sensor nodes are powered by batteries. Most of the sensor node manufacturers use batteries which are not interchangeable easily due to varying form factors and rated voltages. When a sensor node uses a battery form factor not easily available in market or too costly without providing any performance increase (increased capacity in mAh), it results in vendor lock-in and supply chain issues. This is not a desirable scenario for large scalable systems. Note that sometimes the cost of battery is more than the cost of whole LoRa SoC module. Examples of batteries used by manufacturers in different nodes :
 1. Multiple 1.5V batteries in AA form factor
 2. 3.6V Li-SOCL2 batteries in 18505 form factor
 3. 3.6V Li-SOCL2 batteries in 14500 (AA) form factor
 4. 3.7V Li-ion batteries in 18650, 26650, AA and other form factors
- **Sensor configuration tools:** Most of the new commercial LoRaWAN compatible sensor nodes in the market are configured by a mobile application using an NFC-enabled mobile phone or an NFC Pad connected to the PC. The support for the configuration app is not very reliable by the sensor manufacturers. Also, the level of control the user gets over sensor nodes from these mobile applications varies a great deal from manufacturer to manufacturer.

10.2 Challenges with MQTT and cloud integration

- **Security:** There is no support for unsecured connection on port 1883 in cloud-hosted brokers like Azure IoT hub. Also support for security certificates like TLS certificates upload (while using a self-signed certificate) is poor for many gateways with Azure IoT hub and other cloud-hosted MQTT brokers.
- **MQTT connection with cloud servers:** Due to the combination of reasons explained above, sometimes it is very difficult to create a secure connection between the gateway and MQTT servers hosted in the cloud. For example, the Azure IoT hub does not implement the full MQTT standard.
- **MQTT version compatibility:** Gateways sometimes support only older MQTT versions like 3.1.1 but cloud service providers have limited or no support for older MQTT versions.

10.3 Recent advances in LoRa/LoRaWAN technology

This section discusses some of the recent advances in research and development of subsystems related LoRaWAN based WSNs.

- **Chips:** With new Semtech's LR1110 and L1121 series chips, Semtech now offers S-band (Satellite) connectivity with the Semtech cloud. New LR1110 has WiFi on SoC as well. This has great number of applications in asset tracking and Geo-location. Note that this Geo-location is not reliant on GPS. SoC offers Geo-location based on Wi-Fi lookup database and GNSS solvers.
- **Security** Security has improved a lot with new chips offering full hardware AES-128 cryptographic engine.
- **Join Server:** Semtech now offers LoRa join server on the cloud.

10.4 Design principles

This section summarizes some of the design principles that were learned through testing and experimentation during this research. When designing a WSN, it is important to pay close attention to these key ideas:

- Keep the complexity of the overall network manageable.
- Reduce the operational costs of the network as it scales.
- Ensure relatively simple device management for a large variety of heterogeneous sensor nodes by grouping similar sensors under one application.
- Make it easy to modify the existing network configuration at various levels.
- Simplify fault detection and troubleshooting process for technical problems in end devices, gateways, and servers.
- Ensure that a fault in one part of the network does not disrupt the entire network's operation.

A.1 Terminology and concepts in LoRa Communication

1. **Free Space Loss:** The signal between sender and receiver loses power as the distance gets larger. This is called free space loss.
2. **Interference:** Reflected radio waves from buildings and obstacles can interfere with the direct signal.
3. **Multipath:** A propagation phenomenon when a transmitted signal reaches the receiver through more than one path due to reflection from buildings, mountains or other geographic environmental conditions. It can lead to constructive or destructive interference of signal which can result into rapid changes/fading of signal strength over a small distance.
4. **Line of sight propagation:** When the Radio waves travels directly from sender to receiver without any obstacle.
5. **Fresnel Zone:** It is roughly an ellipsoid shaped body around the direct line of sight path between the sender and receiver. Any obstacle between within Fresnel zone will weaken the transmitted signal. Avoid object in Fresnel zone by placing antenna at higher location. If there are obstacles near the path within Fresnel zone, radio waves reflecting from the path arrive out of phase with the signal and reduce the power of the received signal considerably.
6. **Received signal Strength Indication:** RSSI is the power of signal at which receiver receives the signal. It is also called Rx Power.
7. **Receiver sensitivity:** Rx sensitivity is the minimum power threshold at which receiver can demodulate the signal.
8. **Link Margin:** The difference between Rx sensitivity and RSSI is called link margin. It should be positive for the receiver to demodulate the signal. LoRa receivers have Rx sensitivity of around -148 dBm depending on SF and BW.

9. **Forward Error correction:** FEC is the process where redundant error correction bits are added to the actual data to be transmitted. These redundant bits helps in restoring data when data gets corrupted by interference.

LoRaWAN Network Equipment and Development Tools

This section contains a list of LoRaWAN equipment suppliers. Some of these products and services have been used for the prototyping.

B.1 OEM and Service providers

- The Things Industry: For cloud services, extensive documentation on LoRaWAN, developer tools, code repositories LoRaWAN gateways
- Dragino: For LoRaWAN sensors, gateways, development boards, LoRa modules and development kits.
- Heltec Automation: Development boards and gateways
- Milesight: Gateways, sensors, configuration tools and apps
- IMbuildings: People counter sensors for smart buildings
- RAK Wireless technology: Gateways, sensors, development platforms, device management via cloud
- ELSYS: Indoor environment monitoring sensors, temperature humidity sensors

B.2 LoRa Development Boards and Modules

- Heltec LoRa 32 V2 board
- The Things UNO development board
- Pycom LoPy4 development board
- LILYGO TTGO T-Beam
- Cube Cell LoRa development boards

- STM32 Nucleo P-NUCLEO-LRWAN2
- IMST iC880A LoRa concentrator
- Dragino LoRa GPS HAT
- Dragino LoRaWAN concentrator PG1301

B.3 Radio Modules and LoRa chips

- Microchip RN2483 module
- STM32 WL55JCI7 (STM32 WL series chips)
- Semtech SX127x series chips
- Semtech SX130x series chips
- Semtech LR1110, LR1121 series chips
- Murata CMWX1ZZABZ module

B.4 Gateways and Nodes

- RAK831 PILOT Gateway — outdoor
- Dragino DLOS8N — outdoor
- RAK 7268C Edge V2 gateway — indoor
- Milesight UG65 gateway — indoor
- MultiTech Conduit IP67 Base Station — outdoor
- Kerlink Wirnet gateways — outdoor

- [1] Khaled Q Abdelfadeel, Victor Cionca, and Dirk Pesch. Fair adaptive data rate allocation and power control in lorawan. In *2018 IEEE 19th International Symposium on "A World of Wireless, Mobile and Multimedia Networks"(WoWMoM)*, pages 14–15. IEEE, 2018.
- [2] Andrew Banks and Rahul Gupta. *mqtt-v3.1.1*. OASIS MQTT Technical Committee, 2014. Accessed: 08.02.2023.
- [3] Jean-Paul Bardyn, Thierry Melly, Olivier Seller, and Nicolas Sornin. Iot: The era of lpwan is starting now. *ESSCIRC Conference 2016: 42nd European Solid-State Circuits Conference*, pages 25–30, 2016.
- [4] Eddas Bertrand-Martínez, Phelipe Dias Feio, Vagner de Brito Nascimento, Billy Pinheiro, and Antônio Abelém. A methodology for classification and evaluation of iot brokers. In *LANOMS*, 2019.
- [5] Bharat S Chaudhari and Marco Zennaro. *LPWAN Technologies for IoT and M2M Applications*. Academic Press, 2020.
- [6] European Telecommunications Standards Institute. *ETSI EN 300 220-2 V3.2.1*, 2018.
- [7] Gaston C Hillar. *MQTT Essentials-A lightweight IoT protocol*. Packt Publishing Ltd, 2017.
- [8] Gaston C Hillar. *Hands-On MQTT Programming with Python: Work with the lightweight IoT protocol in Python*. Packt Publishing Ltd, 2018.
- [9] Steven J Johnston, Philip J Basford, Florentin MJ Bulot, Mihaela Apetroaie-Cristea, Natasha HC Easton, Charlie Davenport, Gavin L Foster, Matthew Loxham, Andrew KR Morris, and Simon J Cox. City scale particulate matter monitoring using lorawan based air quality iot devices. *Sensors*, 19(1):209, 2019.
- [10] Esad Kadusic, Natasa Zivic, Christoph Ruland, and Narcisa Hadzajlic. A smart parking solution by integrating nb-iot radio communication technology into the core iot platform. *Future Internet*, 14(8):219, 2022.
- [11] Heiko Kozirolek, Sten Grüner, and Julius Rückert. A comparison of mqtt brokers for distributed iot edge computing. In *Software Architecture: 14th European Conference, ECSA 2020, L'Aquila, Italy, September 14–18, 2020, Proceedings 14*, pages 352–368. Springer, 2020.

- [12] Valerie Lampkin, Weng Tat Leong, Leonardo Olivera, Sweta Rawat, Nagesh Subrahmanyam, Rong Xiang, Gerald Kallas, Neeraj Krishna, Stefan Fassmann, Martin Keen, et al. *Building smarter planet solutions with mqtt and ibm websphere mq telemetry*. IBM Redbooks, 2012.
- [13] Shengyang Li, Usman Raza, and Aftab Khan. How agile is the adaptive data rate mechanism of lorawan? In *2018 IEEE global communications conference (GLOBECOM)*, pages 206–212. IEEE, 2018.
- [14] LoRa Alliance. *LoRa Specification*, 2015.
- [15] LoRa Alliance. *LoRaWAN 1.1 Specification*, 2017.
- [16] LoRa Alliance. *RP002-1.0.3 LoRaWAN Regional Parameters*, 2021.
- [17] Riccardo Marini, Walter Cerroni, and Chiara Buratti. A novel collision-aware adaptive data rate algorithm for lorawan networks. *IEEE Internet of Things Journal*, 8(4):2670–2680, 2020.
- [18] Kais Mekki, Eddy Bajic, Frederic Chaxel, and Fernand Meyer. A comparative study of lpwan technologies for large-scale iot deployment. *ICT express*, 5(1):1–7, 2019.
- [19] Microsoft. Microsoft azure iot hub documentation. <https://learn.microsoft.com/en-us/azure/iot-hub/iot-concepts-and-iot-hub>. Accessed: 17.03.2022.
- [20] Nitin Naik. Choice of effective messaging protocols for iot systems: Mqtt, coap, amqp and http. In *2017 IEEE international systems engineering symposium (ISSE)*, pages 1–7. IEEE, 2017.
- [21] Tim Pulver. *Hands-On Internet of Things with MQTT: Build connected IoT devices with Arduino and MQ Telemetry Transport (MQTT)*. Packt Publishing Ltd, 2019.
- [22] Semtech Corporation. *AN1200.22 LoRa Modulation Basics*, 2015. Accessed: 02.09.2022.
- [23] Semtech Corporation. *An In-depth Look at LoRaWAN Class A Devices*, 2019. Accessed: 28.03.2023.
- [24] Semtech Corporation. *An In-depth Look at LoRaWAN Class B Devices*, 2019. Accessed: 28.03.2023.
- [25] Semtech Corporation. *LoRa and LoRaWAN: A Technical Overview*, 2019. Accessed: 02.09.2022.
- [26] Semtech Corporation. *Understanding the LoRa Adaptive Data Rate*, 2019. Accessed: 30.03.2023.
- [27] Semtech Corporation. *SX1272/3/6/7/8 LoRa Modem Design Guide*, 2020. Accessed: 02.09.2022.
- [28] Ritesh Kumar Singh, Priyesh Pappinisseri Puluckul, Rafael Berkvens, and Maarten Weyn. Energy consumption analysis of lpwan technologies and lifetime estimation for iot application. *Sensors*, 20(17):4794, 2020.
- [29] Joachim Tapparel, Orion Afisiadis, Paul Mayoraz, Alexios Balatsoukas-Stimming, and Andreas Burg. An open-source lora physical layer prototype on gnu radio. In *2020 IEEE 21st International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*, pages 1–5. IEEE, 2020.
- [30] Lorenzo Vangelista. Frequency shift chirp modulation: The lora modulation. *IEEE Signal Processing Letters*, 24(12):1818–1821, 2017.